

Actor.cpp

```
// Actor.cpp: implementation of the CActor class.
//
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Actor.h"
#include "ResMan.h"
#include "Behavior.h"
#include "UC2Ani/Bubble.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

extern CResMan gResMan;
const DWORD LONGTIMELATER = 3600000L;    // 1 hour later
const int BUBBLEH=10;
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////

CActor::CActor()
{
    m_pBeh          = NULL;
    m_bMoving       = FALSE;
    m_nRepeat       = 0;
    m_nColorSet     = 0;
    m_bVoice        = FALSE;
    m_pBB           = new CBubble;
    // for movement
    m_ptDestEarth   = CPoint(0, 0);
    m_nDestElev     = 0;
    m_bForward      = TRUE;
    m_pShadow       = NULL;    // CSprite
    m_pTM           = NULL;
}

CActor::~CActor()
{
    if (m_pShadow)
        delete m_pShadow;
    // CStage::DeleteActor will delete bubble

```


Actor.cpp

```
// if (m_pBB)
// {
//     if (m_pStage)
//         m_pStage->GetBubbleList()->Remove(m_pBB);    //
// remove this bubble
//     delete m_pBB;
// }

BOOL CActor::SetBehavior(const int bi, const BOOL bAlarm)
{
    if (m_bMoving)
    {
        TRACE("CActor::SetBehavior() - Actor is moving...\n");
        return FALSE;
    }
    m_mi.SetBehavior(bi);
    m_pBeh = gResMan.GetActorBehavior(m_mi.GetCharID(), bi);
    if (!m_pBeh)
    {
        TRACE1("Invalid behavior index %d\n", bi);
        return FALSE;
    }
    m_ai = 0;
    m_bMoving = ((bi >= AB_WALKF) && (bi <= AB_RES_MOVE2));
    m_nRepeat = m_pBeh->GetRepeat();    // save original repeat
count
    if (bAlarm)
        SetAlarmTick();                // Respond immediately
    return TRUE;
}

// Basic behavior (stance) needed to be done after any behavior
BOOL CActor::BasicStance(const BOOL bChanging)
{
    switch (m_mi.GetState() & AS_MASK)
    {
    case AS_STAND:
        if (bChanging)
            return SetBehavior((m_mi.GetState() & DA_FORWARD)
? AB_STANDINGF : AB_STANDINGB);
        else
            return SetBehavior((m_mi.GetState() & DA_FORWARD)
? AB_STANDF : AB_STANDB);
    case AS_MORPH:
        if (bChanging)
```

```

        return SetBehavior((m_mi.GetState() & DA_FORWARD)
? AB_MORPHINGF : AB_MORPHINGB);
        else
            return SetBehavior((m_mi.GetState() & DA_FORWARD)
? AB_MORPHF : AB_MORPHB);
        case AS_DOZE:
            return SetBehavior((m_mi.GetState() & DA_FORWARD) ?
AB_DOZEF : AB_DOZEB);
    }
    TRACE("CActor - Invalid State %xh\n", m_mi.GetState());
    return FALSE;
}

//=====
// OnIdle
// Returning TRUE means we should render
BOOL CActor::HeartBeat(const DWORD dwCurTick)
{
    if (!m_pBeh)
    {
        TRACE("CActor(%x)::HeartBeat: m_pBeh = NULL\n", this);
        SetAlarmTick(dwCurTick + LONGTIMELATER);
        return FALSE;
    }
    CBehavior::Cell* pC = m_pBeh->GetCell(m_ai); // includes
range check
    if (!pC) // One behavior ended
    {
        TRACE("CActor(%x)::HeartBeat: Behavior(%d) done.\n",
this, m_pBeh->GetID());
        m_ai = 0; // wrap index

        if (m_bMoving)
        {
            SetElevation(m_nDestElev);
            MoveToEarth(m_ptDestEarth);
            SetZByEarth();
            ASSERT(m_pBB);
            if (m_pBB->IsShown())
                AdjustBubblePosition();
        }
        // if repeat count is infinite(=0) or left, continue
this action
        if (m_pBeh->GetRepeat() == 0) // 0 means infinite loop
            return TRUE;
        if (--m_nRepeat <= 0)

```

```

    {
        m_bMoving = FALSE;
        if (m_mi.GetBehavior() >= AB_WALKF) //
otherwise it is a basic stance
            return BasicStance(FALSE); // do not show
changing cuts
        SetAlarmTick(dwCurTick + LONGTIMELATER);
    }
    return FALSE;
}

SetCell(pC->nID); // update the view with the current
cell(frame)
SetImOp((m_mi.GetState() & DA_RIGHT) ? pC->wIO : (pC->wIO |
IMAGE_FLIP)); // calls m_pNotifyObj->Change(...)
if (pC->nSI >= 0)
    gResMan.PlayWave(pC->nSI);
if (pC->ptDisp.x || pC->ptDisp.y)
{
    WORD wDA = m_mi.GetState() & DA_MASK;
    if (!m_bForward) // Reverse direction
    {
        if (wDA <= DA_FL) wDA <= 2;
        else wDA >= 2;
    }
    switch (wDA)
    {
        case DA_FR: MoveBy(-pC->ptDisp.x, +pC->ptDisp.y);
break;
        case DA_FL: MoveBy(+pC->ptDisp.x, +pC->ptDisp.y);
break;
        case DA_BR: MoveBy(+pC->ptDisp.x, -pC->ptDisp.y);
break;
        case DA_BL: MoveBy(-pC->ptDisp.x, -pC->ptDisp.y);
break;
    }
    if (GetElevation()) // && ((wDA & DA_FORWARD) == wDA))
// increasing z-order
    {
        if (m_ai == 1) // 0,1,2,3, half the way
            SetZ(GetZ() - 16); // half the height of
the tile
    }
    else
    {
        SetZByEarth();
    }
}

```

```

        ASSERT(m_pBB);
        if (m_pBB->IsShown())
            AdjustBubblePosition();
    }

    // Set Alarm time to be called for the next frame
    SetAlarmTick(dwCurTick + (DWORD)pC->nTicks * GetMSPT());
    m_ai++;
    return TRUE;    // call RenderAndDrawDirtyList() in
CStage::TickAll()
}

// FR(1) -> FL(2) -> BR(4) -> BL(8) -> FR(1)
// counter-clockwise
WORD CActor::GetTurnNextDA() const
{
    WORD wDA = m_mi.GetState() & DA_MASK;
    ASSERT(wDA);
    ASSERT(DA_FR == 0x0001);
    ASSERT(DA_BL == 0x0008);
    wDA <<= 1;
    if (wDA > DA_BL)
        wDA = DA_FR;
    return wDA;
}

WORD CActor::GetTurnPrevDA() const
{
    WORD wDA = m_mi.GetState() & DA_MASK;
    ASSERT(wDA);
    ASSERT(DA_FR == 0x0001);
    ASSERT(DA_BL == 0x0008);
    wDA >>= 1;
    if (wDA < DA_FR)
        wDA = DA_BL;
    return wDA;
}

BOOL CActor::Act(const int nCmd)
{
    if (m_bMoving)
    {
        TRACE("CActor::Act() - Actor is moving...\n");
        return FALSE;
    }
    switch (nCmd)
    {

```

```

// Position Move
// case CMD_MOVEF:
// case CMD_MOVEB:
// State Change
    case CMD_STAND:      return SetSA(AS_STAND);
    case CMD_MORPH:      return SetSA(AS_MORPH);
    case CMD_DOZE: return SetSA(AS_DOZE);
    case CMD_TURNL:      return SetDA(GetTurnNextDA(), FALSE);
    case CMD_TURNR:      return SetDA(GetTurnPrevDA(), FALSE);
// Actions
    case CMD_CHAT:
    case CMD_HELLO:
        if ((m_mi.GetState() & AS_MORPH) == AS_MORPH)
            return TRUE;    // No behavior will be shown for
morphed actor.
        return SetBehavior((m_mi.GetState() & DA_FORWARD) ?
AB_CHAT : AB_WIGGLEB);
    case CMD_ENTER:
    case CMD_EXIT:
    case CMD_SMILE:
    case CMD_MAD:
    case CMD_CRY:
// case CMD_SCRATCH:
    case CMD_PICK:
    case CMD_SPECIAL:
        if ((m_mi.GetState() & AS_MORPH) == AS_MORPH)
            return TRUE;    // No behavior will be shown for
morphed actor.
    case CMD_SCRATCH:
        return SetBehavior(AB_CHAT + nCmd - CMD_CHAT);
    case CMD_PUNCH:
        return SetBehavior((m_mi.GetState() & DA_FORWARD) ?
AB_PUNCHF : AB_PUNCHB);
    case CMD_BEATEN:
        return SetBehavior((m_mi.GetState() & DA_FORWARD) ?
AB_BEATENF : AB_BEATENB);
    }
    return FALSE;
}

BOOL CActor::MoveTo(const CPoint& ptDest, const int nEA, const
BOOL bForward)
{
    if (m_bMoving)
    {
        TRACE("CActor::MoveTo() - Actor is moving...\n");
        return FALSE;
    }

```

```

    }
    m_ptDestEarth = ptDest;
    m_nDestElev    = nEA;
    m_bForward     = bForward;
    switch (m_mi.GetState() & AS_MASK)
    {
    case AS_STAND:
    case AS_DOZE:
        if (nEA == GetElevation())
        {
            return SetBehavior((m_mi.GetState() & DA_FORWARD)
? AB_WALKF : AB_WALKB);
        }
        else if (nEA > GetElevation())
        {
            if (!bForward)
            {
                SetBehavior(AB_SCRATCH);
                return FALSE; // can't go up backward
            }
            return SetBehavior((m_mi.GetState() & DA_FORWARD)
? AB_UPF : AB_UPB);
        }
        else
        {
            return SetBehavior((m_mi.GetState() & DA_FORWARD)
? AB_DOWNF : AB_DOWNB);
        }
        case AS_MORPH:
            return SetBehavior((m_mi.GetState() & DA_FORWARD) ?
AB_MORPHWALKF : AB_MORPHWALKB);
        }
        return FALSE;
    }

void CActor::Chat(const CString& strText)
{
    CString strBubble(strText);
    if (strBubble.GetLength() > gResMan.GetBubbleTextLimit())
    {
        strBubble.GetBufferSetLength(gResMan.GetBubbleTextLimit());
        strBubble.ReleaseBuffer();
        strBubble += "...";
    }

    CRect rcA; // This Actor's Rectangle

```

```

    GetRect(rcA);
    m_pBB->SetKind(m_mi.GetBubbleKind());
    WORD wDA = m_mi.GetState() & DA_MASK;
    if ((wDA == DA_FL) || (wDA == DA_BR))
    {
        m_pBB->SetTextAlign(TA_RIGHT);
        m_pBB->TextOut(rcA.right, rcA.top-BUBBLEH, strBubble);
    }
    else
    {
        m_pBB->SetTextAlign(TA_LEFT);
        m_pBB->TextOut(rcA.left, rcA.top-BUBBLEH, strBubble);
    }
    m_pBB->SetZ(-GetEarthPointY());
    m_pBB->Show(TRUE);

    // Set duration somewhat proportional to the length of the
    talk
    DWORD dwDelay = gResMan.GetBubbleTime() +
    strBubble.GetLength() * 80;
    // if (dwDelay > 8000) // in 1/10 sec unit
    //     dwDelay = 8000;
    m_pBB->SetAlarmTick(::GetTickCount() + dwDelay);

    if (m_bMoving)
        return;
    if (strText.Find("¾û¾û") >= 0) Act(CMD_CRY);
    else if ((strText.Find("ÇIÇI") >= 0) ||
             (strText.Find("ÈÈÈÈ") >= 0)) Act(CMD_SMILE);
    else if (strText.Find("´çÈ²") >= 0)
        Act(CMD_SCRATCH);
    else if (strText.Find("¾È³ ç") >= 0)
        Act(CMD_HELLO);
    else if (strText.Find("¾¾¹ß") >= 0) Act(CMD_MAD);
    else Act(CMD_CHAT);
    // includes range check
    SetAlarmTick(); // Force immediate action
}

void CActor::AdjustBubblePosition()
{
    CRect rcA;
    GetRect(rcA);
    if (m_pBB->GetTextAlign() == TA_LEFT)
        m_pBB->MovePosition(rcA.left, rcA.top-BUBBLEH);
    else
        m_pBB->MovePosition(rcA.right, rcA.top-BUBBLEH);
}

```

```

        m_pBB->SetZ(-GetEarthPointY());
    }

// virtual
void CActor::Render(CDIB* pDIB, const CRect* pClipRect)
{
    // The sprite Shadow need not have z-order
    // since it is always rendered with CActor
    if (m_pShadow)
        m_pShadow->Render(pDIB, pClipRect);
    CPhasedSprite::Render(pDIB, pClipRect);
}

// virtual in CSprite
void CActor::SetLT(const int x, const int y)
{
    CPhasedSprite::SetLT(x, y);
    if (m_pTM)
    {
        m_mi.SetTileID(m_pTM-
>GetNearestTileIndex(GetEarthPoint()));
    }
    if (m_pShadow)
    {
        CSize sE(GetEarth()); // of this actor
        CSize sS(m_pShadow->GetWidth()/2, m_pShadow-
>GetHeight()/2);
        m_pShadow->SetLT(x + sE.cx - sS.cx, y + sE.cy - sS.cy
+ 2);
    }
}

// virtual
void CActor::SetImOp(const WORD wImOp)
{
    CPhasedSprite::SetImOp(wImOp);
    if (m_pShadow)
        m_pShadow->SetImOp(wImOp);
}

void CActor::InitState(CMemberInfo& mi)
{
    m_mi = mi;
    m_pBB->SetKind(m_mi.GetBubbleKind());
}

void CActor::GetMemberInfo(CMemberInfo& mi)

```


Actor.cpp

```
{
    mi = m_mi;
}
```

BOOL CActor::IsMemberIgnored() const

```
{
    PICS_MEMBER pM = m_mi.GetMember();
    if (pM)
    {
        BOOL bIgnored = (pM->HrIsMemberIgnored() == NOERROR);
        pM->Release();
        return bIgnored;
    }
    return TRUE;    // Ignored? Ya~
}
```

BOOL CActor::IsMemberHost() const

```
{
    PICS_MEMBER pM = m_mi.GetMember();
    if (pM)
    {
        BOOL fRet = (pM->HrIsMemberHost() == NOERROR);
        pM->Release();
        return fRet;
    }
    return FALSE;
}
```

```

// Actor.h: interface for the CActor class.
//
///////////////////////////////////////////////////////////////////
#ifndef AFX_ACTOR_H__C78B9066_A908_11D1_80E2_080009B9F339__INCLUDED_
#define AFX_ACTOR_H__C78B9066_A908_11D1_80E2_080009B9F339__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#include "UC2Ani/PhSprite.h"
#include "UC2Messages.h"
#include "TileMap.h" // DA_...
#include "MemberInfo.h"
class CBehavior;
class CBubble;
class CSprite;
class CActor : public CPhasedSprite
{
public:
    CActor();
    virtual ~CActor();

// Attributes
    int                GetCharID() const    { return m_mi.GetCharID(); }
    int                GetColorSet() const { return m_nColorSet; }
    CBubble*           GetBubble() const    { return m_pBB; }
    WORD               GetState() const     { return m_mi.GetState(); }
    WORD               GetDA() const { return (m_mi.GetState() & DA_MASK); }
    WORD               GetTurnNextDA() const;
    WORD               GetTurnPrevDA() const;
    BOOL               IsMoving() const     { return m_bMoving; }
    void               SetShadow(CSprite* pS) { m_pShadow = pS; }
    BOOL               IsVoice() const      { return m_bVoice; }
    PICS_MEMBER        GetMember()          { return m_mi.GetMember(); }

    const CString* GetNick() const          { return m_mi.GetNick(); }
    void             GetMemberInfo(CMemberInfo& mi);
    int             GetGender() const       { return m_mi.GetSex(); } // 0:
    Male, 1: Female, 2: Child, ...
    BOOL            IsMemberIgnored() const;
    BOOL            IsMemberHost() const;

// Operations
    virtual void     SetLT(const int x, const int y); // For Shadow
    virtual void     SetImOp(const WORD wImOp); // For Shadow
    virtual void     Render(CDIB* pDIB, const CRect* pClipRect=NULL); // For Shadow
    void             SetTileMap(CTileMap* pTM) { m_pTM = pTM; }

```

Actor.h

```

        void                SetCharID(const int nCharID)        {
m_mi.SetCharID(nCharID); }
        void                SetColorSet(const int nCS)    { m_nColorSet = nCS; }
        BOOL                SetBehavior(const int bi, const BOOL bAlarm=TRUE);
        BOOL                SetState(const WORD wState, const BOOL bChanging=TRUE)
                        { m_mi.SetState(wState); return BasicStance(bChanging);
    }

        BOOL                SetSA(const WORD wSA, const BOOL bChanging=TRUE)
                        { m_mi.SetSA(wSA); return BasicStance(bChanging); }
        BOOL                SetDA(const WORD wDA, const BOOL bChanging=TRUE)
                        { m_mi.SetDA(wDA); return BasicStance(bChanging); }
        virtual BOOL        HeartBeat(const DWORD dwCurTick);    // ticker
        BOOL                Act(const int nCmd);
        BOOL                MoveTo(const CPoint& ptDest, const int nEA, const BOOL
bForward=TRUE);
        void                Chat(const CString& strText);
        // Attach the PICS member interface object to this CActor object
        BOOL                Attach(PICS_MEMBER pMem)    { return
m_mi.SetMember(pMem); }
        void                InitState(CMemberInfo& mi);
        CMemberInfo m_mi;
protected:
        BOOL                BasicStance(const BOOL bChanging=TRUE);    // Show
changing cuts
        void                AdjustBubblePosition();
        BOOL                m_bMoving;    // Actor is moving, do not interrupt
        int                m_nColorSet;
        CBehavior*          m_pBeh;        // pointer to the CBehavior obj
        int                m_nRepeat;    // current Repeat count for behavior
        CBubble*            m_pBB;        // holds an instance for the CBubble
//        CQueue            m_ActionQ;
        BOOL                m_bVoice;
// for movement
        CPoint              m_ptDestEarth;
        int                m_nDestElev;
        BOOL                m_bForward;
        CSprite*            m_pShadow;
        CTileMap*           m_pTM;
};
#endif //
!defined(AFX_ACTOR_H__C78B9066_A908_11D1_80E2_080009B9F339__INCLUDED_)

```

```

//-----
//      CBaseChannel class
//
//      Copyright (C) 1996 Microsoft Corporation
//      All rights reserved.
//
//      Modification for MFC
//      (C) Programmed by Kim,
//      unichat Media Lab
//      Information Technology Institue
//
//-----
#include "stdafx.h"
#include "BaseChan.h"

CBaseChannel::CBaseChannel()
{
    TRACE0("CBaseChannel::CBaseChannel()\n");
    m_picsChannel      = NULL;
    m_pThread          = NULL;
    m_hThread          = NULL;
    m_fGotMemList      = FALSE;
    m_nPICSChanRef     = 0;
}

CBaseChannel::~CBaseChannel()
{
    TRACE0("CBaseChannel::~CBaseChannel()\n");
    CleanUp();
}

//      Save the channel pointer returned by ChatSock, then start a message
BOOL CBaseChannel::FInit(PICS_CHANNEL pChannel)
{
    ASSERT(pChannel);

    WaitForMsgThread();

    SetChannel(pChannel);

    if (!FStartMessageThread())
    {
        CleanUp();
        return FALSE;
    }
    return TRUE;
}

//      Resets the object so it can be restarted with a new channel.
BOOL CBaseChannel::FReInit()
{
    CleanUp();

    return TRUE;
}

//      Release our channel pointer.

```

```

void CBaseChannel::CleanUp()
{
    m_fGotMemList      = FALSE;
    WaitForMsgThread();
    if (m_picsChannel)
    {
        m_picsChannel->Release();
        m_picsChannel = NULL;
    }
}

// Wait for our message thread to terminate; we need to do this
// so we can be sure the thread is cleaned up so we can exit or reinitialize
// the object.
void CBaseChannel::WaitForMsgThread()
{
    if (m_hThread)
    {
        TRACE("CBaseChannel - ::WaitForSingleObject(0x%lx, 5000L);\n",
m_hThread);
        DWORD dwRc = ::WaitForSingleObject(m_hThread, 5000L);
        switch (dwRc)
        {
            case WAIT_OBJECT_0:
                TRACE0("Thread: WAIT_OBJECT_0\n");
                break;
            case WAIT_ABANDONED:
                TRACE0("Thread: WAIT_ABANDONED\n");
                break;
            case WAIT_TIMEOUT:
                TRACE0("Thread Hung! Deleting...\n");
                //::MessageBox(NULL, "Error: Cannot terminate channel
thread.\n", "CBaseChannel", MB_OK);
                delete m_pThread;
                break;
            case WAIT_FAILED:
                PrintWin32Error("Thread: WAIT_FAILED ");
                break;
            default:
                TRACE0("Thread Hung!\n");
                break;
        }
        //::CloseHandle(m_hThread); // CWinThread may have done this
for us.
        m_pThread = NULL; // CWinThread auto-deleted
        m_hThread = NULL;
    }
}

////////////////////////////////////
// CBaseChannel accessors

// Don't miss calling Release() after using the returned pointer.
PICS_CHANNEL CBaseChannel::PChannel()
{
    if (m_picsChannel)
        m_picsChannel->AddRef();
}

```

```

        return m_picsChannel;
    }

    BOOL CBaseChannel::FInChannel()
    {
        return (NULL != m_picsChannel);
    }

    // FLeave() will Release()
    void CBaseChannel::SetChannel(PICS_CHANNEL pics)
    {
        ASSERT(!m_picsChannel);

        pics->AddRef();
        m_picsChannel = pics;
    }

    // Get Channel name
    char* CBaseChannel::SzName()
    {
        ASSERT(m_picsChannel);

        BYTE* pb;
        BOOL fAnsi;
        HRESULT hr = m_picsChannel->HrGetName(&pb, &fAnsi);

        // For simplicity, ignore non-ANSI...
        if (FAILED(hr) || !fAnsi)
            return NULL;

        return (char*)pb;
    }

    // Returns the name of the specified member.
    char* CBaseChannel::SzMemName(PICS_MEMBER pMember)
    {
        ASSERT(pMember);

        BYTE* pb;
        BOOL fAnsi;
        HRESULT hr = pMember->HrGetName(&pb, &fAnsi);

        // For simplicity, ignore non-ANSI...
        if (FAILED(hr) || !fAnsi)
            return NULL;

        return (char*)pb;
    }

    char* CBaseChannel::SzTopic()
    {
        ASSERT(m_picsChannel);

        BYTE* pb;
        BOOL fAnsi;
        HRESULT hr = m_picsChannel->HrGetTopic(&pb, &fAnsi);
    }

```

```

        // For simplicity, ignore non-ANSI...
        if (FAILED(hr) || !fAnsi)
            return NULL;

        return (char*)pb;
    }

    // Is the specified member myself?
    BOOL CBaseChannel::FIsMemberMe(PICS_MEMBER pMember)
    {
        ASSERT(m_picsChannel);
        // ASSERT(pMember);

        PICS_MEMBER pMem;
        if (FAILED(m_picsChannel->HrGetMe(&pMem)))
        {
            return FALSE;
        }
        BOOL fRet = (pMember == pMem);
        if (pMem)
        {
            pMem->Release();
        }
        return fRet;
    }

    BOOL CBaseChannel::FAmIHost()
    {
        ASSERT(m_picsChannel);

        PICS_MEMBER pMem;
        if (FAILED(m_picsChannel->HrGetMe(&pMem)))
        {
            return FALSE;
        }
        if (pMem)
        {
            BOOL fRet = (NOERROR == pMem->HrIsMemberHost());
            pMem->Release();
            return fRet;
        }
        return FALSE;
    }

    //////////////////////////////////////
    // CBaseChannel operations

    BOOL CBaseChannel::FSendAnsiText(char* szText)
    {
        ASSERT(m_picsChannel);

        HRESULT hr = m_picsChannel->HrSendTextA(szText);
        if (FAILED(hr))
        {
            FOnChannelError(hr);    // virtual function call
            return FALSE;
        }
    }

```

```

    }
    return TRUE;
}

BOOL CBaseChannel::FSendData(BYTE* pbData, DWORD dwcb)
{
    ASSERT(m_picsChannel);

    HRESULT hr = m_picsChannel->HrSendData(pbData, dwcb);
    if (FAILED(hr))
    {
        FOnChannelError(hr);    // virtual function call
        return FALSE;
    }
    return TRUE;
}

BOOL CBaseChannel::FWhisperTo(PICS_MEMBER pM, char* szText)
{
    ASSERT(m_picsChannel);

    HRESULT hr = m_picsChannel->HrSendTextListA(szText, &pM, 1);
    if (FAILED(hr))
    {
        FOnChannelError(hr);    // virtual function call
        return FALSE;
    }
    return TRUE;
}

BOOL CBaseChannel::FSendInvite(PCS_INVITEINFO piInfo)
{
    ASSERT(m_picsChannel);

    HRESULT hr = m_picsChannel->HrSendInviteA(piInfo);
    if (FAILED(hr))
    {
        FOnChannelError(hr);    // virtual function call
        return FALSE;
    }
    return TRUE;
}

DWORD CBaseChannel::DwUserCount()
{
    ASSERT(m_picsChannel);

    DWORD cUser;
    HRESULT hr = m_picsChannel->HrGetUserCount(&cUser);
    if (FAILED(hr))
    {
        FOnChannelError(hr);    // virtual function call
        return 0;
    }
    return cUser;
}

```



```

DWORD CBaseChannel::DwType()
{
    ASSERT(m_picsChannel);

    DWORD dwMode;
    HRESULT hr = m_picsChannel->HrGetType(&dwMode);
    if (FAILED(hr))
    {
        FonChannelError(hr);    // virtual function call
        return 0;
    }
    return dwMode;
}

BOOL CBaseChannel::FSetTopic(char* szTopic)
{
    ASSERT(m_picsChannel);

    HRESULT hr = m_picsChannel->HrSetTopicA(szTopic);
    if (FAILED(hr))
    {
        FonChannelError(hr);    // virtual function call
        return FALSE;
    }
    return TRUE;
}

BOOL CBaseChannel::FLeave()
{
    ASSERT(m_picsChannel);

    if (m_picsChannel)
    {
        if (FAILED(m_picsChannel->HrLeave(FALSE)))
            return FALSE;

        WaitForMsgThread();
        m_picsChannel->Release();
        if (m_nPICSCChanRef)
            m_picsChannel->Release();
        m_picsChannel = NULL;
    }
    return NOERROR;
}

//      Waits for a message to arrive on the message queue.
// Calling FLeave on the channel will cause this method to return FALSE
// immediately.
// Dispatches the received message using the overrideable virtual methods of
// CBaseChannel.
BOOL CBaseChannel::FWaitForMessage()
{
    if (!m_picsChannel)
        return FALSE;
    m_picsChannel->AddRef();
    count so that we can be sure that                                // increase the ref

```

```

TRACE0("m_picsChannel->AddRef();\n");      // the channel object doesn't
go away until this function ends...
m_nPICSChanRef++;

PCS_MSGBASE pcsMsg;
while (SUCCEEDED(m_picsChannel->HrWaitForMsg(&pcsMsg, INFINITE)))
{
    DebugMessageType("=> CBaseChannel::FWaitForMessage", pcsMsg-
>csMsgType);
    switch (pcsMsg->csMsgType)
    {
    default:
        FUnknownMessage(pcsMsg);
        break;

    case CSMSG_TYPE_ERROR:
        {
            PCS_ERROR pErr = MSGBASE_TO_MSG(pcsMsg, PCS_ERROR);
            FOnChannelError(pErr->hr);      // virtual function call
        }
        break;

    case CSMSG_TYPE_ADDMEMBER:
        {
            PCS_MSGMEMBER pAddMsg = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGMEMBER);
            FOnAddMember(pAddMsg);          // virtual function call
        }
        break;

    case CSMSG_TYPE_GOTMEMLIST:
        {
            m_fGotMemList = TRUE;
            FOnGotMemList();
        }
        break;

    case CSMSG_TYPE_DELMEMBER:
        {
            PCS_MSGMEMBER pDelMsg = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGMEMBER);
            FOnDelMember(pDelMsg);          // virtual function call
        }
        break;

    case CSMSG_TYPE_DELCHANNEL:
        {
            PCS_MSGCHANNEL pMsgChan = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGCHANNEL);
            FOnDelChannel(pMsgChan);        // virtual function call
        }
        break;

    case CSMSG_TYPE_MODEMEMBER:
        {
            PCS_MSGMEMBER pModeMsg = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGMEMBER);
            FOnMemberModeChange(pModeMsg);  // virtual function
call

```

```

        }
        break;

    case CSMSG_TYPE_MODECHANNEL:
        FonChannelModeChange(); // virtual function
call
        break;

    case CSMSG_TYPE_TEXT_A:
    {
        PCS_MSG pMsgText = MSGBASE_TO_MSG(pcsMsg, PCS_MSG);
call
        FonAnsiTextMsg(pMsgText); // virtual function
    }
    break;

    case CSMSG_TYPE_DATA:
    {
        PCS_MSG pMsgData = MSGBASE_TO_MSG(pcsMsg, PCS_MSG);
call
        FonDataMsg(pMsgData); // virtual function
    }
    break;

    case CSMSG_TYPE_WHISPERTEXT_A:
    {
        PCS_MSGWHISPER pMsgWhisper = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGWHISPER);
call
        FonAnsiWhisperTextMsg(pMsgWhisper); // virtual function
    }
    break;

    case CSMSG_TYPE_WHISPERDATA:
    {
        PCS_MSGWHISPER pMsgWhisperData = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGWHISPER);
function call
        FonAnsiWhisperDataMsg(pMsgWhisperData); // virtual
    }
    break;

    case CSMSG_TYPE_NEWTOPIC:
        FonNewTopic(); // virtual function call
        break;

    case CSMSG_TYPE_NEWNICK:
    {
        PCS_NEWNICK pNickMsg = MSGBASE_TO_MSG(pcsMsg, PCS_NEWNICK);
        FonNewNick(pNickMsg); // virtual function call
    }
    break;

    // Free the msg
    ::HrFreeMsg(pcsMsg);
}
if (m_picsChannel)

```

```

    {
        m_picsChannel->Release();
        m_nPICSChanRef--;
        TRACE0("m_picsChannel->Release();\n");
    }
    TRACE0("End of CBaseChannel::FWaitForMessage()\n");

    return TRUE;
}

BOOL CBaseChannel::FStartMessageThread()
{
    ASSERT(!m_hThread);

    // m_hThread = ::CreateThread(NULL, 0, DwChannelThreadProc, this, 0,
    &dwID);
    m_pThread = AfxBeginThread(ChannelThreadProc, (LPVOID)this);
    ASSERT(m_pThread);
    m_hThread = m_pThread->m_hThread;    // Save the handle
    // We need to save the handle since the object m_pThread points to may
    already
    // be deleted away. And pass this handle to ::WaitSingleObject.
    TRACE("CBaseChannel created a thread for DwChannelThreadProc [%lx]\n",
        m_pThread->m_nThreadID);
    return (m_pThread != NULL);
}

// static
UINT CBaseChannel::ChannelThreadProc(LPVOID pvData)
{
    ASSERT(pvData);
    CBaseChannel* pbasechannel = (CBaseChannel*)pvData;
    return pbasechannel->FWaitForMessage();
}

```

BaseChan.h

```
//-----
//      CBaseChannel class
//
//      (C) Programmed by Kim,
//unichat Lab
//      Information Technology Institue
//      unichat
//
//      Original Copyright (C) 1996 Microsoft Corporation
//      All rights reserved.
//
//      Modified and trouble-shooted for MFC
//      ::CreateThread ----> CWinThread* AfxBeginThread
//-----
#ifndef _BASECHAN_H
#define _BASECHAN_H

////////////////////////////////////
// class CBaseChannel

//      CBaseChannel is a wrapper around ICSCChannel.
// It provides message-handling code and wrappers around some ICSCChannel
// functionality.
// If you need more functionality from ChatSock, use CBaseChannel::PChannel()
// to obtain a channel interface and then call ChatSock directly.
// Note: if you intend to keep that pointer around in other data structures,
// you should AddRef() it and then Release() it when that particular data
// structure
// goes away. This will make your cleanup logic a lot more robust.

class CBaseChannel : public CObject
{
public:
    CBaseChannel();
    virtual ~CBaseChannel();

    BOOL          FInit(PICS_CHANNEL pChannel);
    BOOL          FReInit();

    PICS_CHANNEL  PChannel();
    BOOL          FInChannel();

    BOOL          FGotMemList() const          { return m_fGotMemList; }

    BOOL          FAmIHost();
    BOOL          FIsMemberMe(PICS_MEMBER pMember);
    char*         SzName();
    char*         SzMemName(PICS_MEMBER pMember);
    char*         SzTopic();

    BOOL          FGotMemberList()              { return
m_fGotMemList; }
    BOOL          FSendAnsiText(char* szText);
    BOOL          FSendData(BYTE* pbData, DWORD dwcb);
    BOOL          FWhisperTo(PICS_MEMBER pM, char* szText);
    BOOL          FSendInvite(PICS_INVITEINFO piInfo);
};
```

BaseChan.h

```

        DWORD          DwUserCount();
        DWORD          DwType();
        BOOL           FSetTopic(char* szText);
        BOOL           FLeave();

        // Overrideables
        virtual BOOL    FOnChannelError(HRESULT hr)
        { return TRUE; }
        virtual BOOL    FOnAddMember(PCS_MSGMEMBER pMsg)
        { return TRUE; }
        virtual BOOL    FOnDelMember(PCS_MSGMEMBER pMsg)
        { return TRUE; }
        virtual BOOL    FOnDelChannel(PCS_MSGCHANNEL pMsg)
        { return TRUE; }
        virtual BOOL    FOnMemberModeChange(PCS_MSGMEMBER pMsg)
        { return TRUE; }
        virtual BOOL    FOnChannelModeChange()
        { return TRUE; }
        virtual BOOL    FOnNewTopic()
        { return TRUE; }
        virtual BOOL    FOnAnsiTextMsg(PCS_MSG pMsg)
        { return TRUE; }
        virtual BOOL    FOnDataMsg(PCS_MSG pMsg)
        { return TRUE; }
        virtual BOOL    FOnAnsiWhisperTextMsg(PCS_MSGWHISPER pMsg)
        { return TRUE; }
        virtual BOOL    FOnAnsiWhisperDataMsg(PCS_MSGWHISPER pMsg)
        { return TRUE; }
        virtual BOOL    FOnNewNick(PCS_NEWNICK pMsg)
        { return TRUE; }
        // and if we got something we don't really handle...
        virtual BOOL    FUnknownMessage(PCS_MSGBASE pMsg)
        { return TRUE; }

protected:
//      friend DWORD __stdcall ChannelThreadProc(PVOID pvData);
        static UINT      ChannelThreadProc(LPVOID pvData);

        void             CleanUp();
        void             SetChannel(PICS_CHANNEL pics);
        BOOL             FWaitForMessage();
        BOOL             FStartMessageThread();
        void             WaitForMsgThread();

        PICS_CHANNEL     m_picsChannel;
        BOOL             m_fGotMemList;
        CWinThread*       m_pThread;
        HANDLE            m_hThread;
        int               m_nPICSChanRef;
};

#ifdef _DEBUG
        void DebugMessageType(LPCSTR t, CSMSG_TYPE c);
        DWORD PrintWin32Error(LPCSTR pszErrorString);
#else
        #define DebugMessageType(t, c) // Do nothing
        #define PrintWin32Error(t)

```

BaseChan.h

#endif

#endif // _BASECHAN_H

```

//-----
//      CBaseSocket class
//
//      (C) Programmed by Kim,
//      unichat Lab
//      Information Technology Institue
//
//
//      Copyright (C) 1996 Microsoft Corporation
//      All rights reserved.
//
//      Modified and trouble-shooted for MFC
//      ::CreateThread ----> CWinThread* AfxBeginThread
//-----
#include "stdafx.h"
#include "BaseSock.h"
#include "resource.h"          // main symbols

//DWORD __stdcall DwSocketThreadProc(PVOID pvData);

CBaseSocket::CBaseSocket()
{
    TRACE0("CBaseSocket::CBaseSocket()\n");
    m_pics                = NULL;
    m_pFactory             = NULL;
    m_pThread              = NULL;
    m_hThread              = NULL;
    m_bCanceled            = FALSE;
    m_nPICSRef             = 0;
}

CBaseSocket::~CBaseSocket()
{
    TRACE0("CBaseSocket::~CBaseSocket()\n");
    CleanUp();
}

//      Close the socket and kill the worker thread.
void CBaseSocket::CleanUp()
{
    if (!m_bCanceled)
        FCloseChatSocket();

    if (m_pFactory)
    {
        m_pFactory->Release();
    }
    if (m_pics)
    {
        m_pics->Release();
    }
    WaitForMsgThread();
}

// We need to make sure to wait for the thread to exit, so we
// can be sure all the resources have been cleaned up.
void CBaseSocket::WaitForMsgThread()

```



```

{
    if (m_hThread)
    {
        TRACE("CBaseSocket - ::WaitForSingleObject(0x%lx, 5000L);\n",
m_hThread);
        DWORD dwRc = ::WaitForSingleObject(m_hThread, 5000L);
        switch (dwRc)
        {
            case WAIT_OBJECT_0:
                TRACE0("Thread: WAIT_OBJECT_0\n");
                break;
            case WAIT_ABANDONED:
                TRACE0("Thread: WAIT_ABANDONED\n");
                break;
            case WAIT_TIMEOUT:
                TRACE0("Thread Hung! Deleting...\n");
                ::MessageBox(NULL, "Error: Cannot terminate socket
thread.\n", "CBaseSocket", MB_OK);
                // delete m_pThread;
                break;
            case WAIT_FAILED:
                PrintWin32Error("Thread: WAIT_FAILED ");
                break;
            default:
                TRACE0("Thread Hung!\n");
                break;
        }
        //::CloseHandle(m_hThread);
        m_pThread = NULL; // CWinThread auto-deleted
        m_hThread = NULL;
    }
}

// Log off and close the socket.
BOOL CBaseSocket::FCloseChatSocket()
{
    if (m_pics)
    {
        TRACE0("CBaseSocket::FCloseChatSocket(): m_pics->HrLogOff()\n");
        // After a successful log off,
        m_pics->HrLogOff();
        // IChatSocket::HrWaitForMsg method will return a CS_E_QUEUEEMPTY
message/
        // You should then terminate your message loop.
        WaitForMsgThread(); // Soomin Kim
        TRACE0("CBaseSocket::FCloseChatSocket(): m_pics-
>HrCloseSocket()\n");
        m_pics->HrCloseSocket(); // A thread in ChatSock.dll exits.
(0x80004601C)
        m_pics->Release();
        if (m_nPICSRef)
        {
            m_pics->Release();
            m_nPICSRef--;
        }
        m_pics = NULL;
    }
}

```

BaseSock.cpp

```

        // Don't set m_pics = NULL here. Since
CBaseSocket::FWaitForMessage() may still
        // be running.
    }
    return TRUE;
}

PICS CBaseSocket::PChatSocket()
{
    ASSERT(m_pics);

    if (m_pics)
        m_pics->AddRef();
    return m_pics;
}

// Initialize the object. We create the socket factory here, so we can
// allow disconnecting during the process of connecting; a connection attempt
// can take a long time, so if the caller wants to cancel the process,
// it can call HrDisconnect() in another thread.
BOOL CBaseSocket::FInit()
{
    // ASSERT(!m_pics);
    if (m_pics)
    {
        TRACE0("CBaseSocket::FInit() - m_pics != NULL\n");
        return FALSE;
    }

    if (m_pFactory)
    {
        m_pFactory->Release();
    }
    TRACE0("CBaseSocket: ::HrCreateChatSocketFactory(IID_CHATSOCKVER1,
&m_pFactory)\n");
    HRESULT hr = ::HrCreateChatSocketFactory(IID_CHATSOCKVER1,
&m_pFactory);
    if (FAILED(hr))
    {
        FOnSocketError(hr); // virtual function call
        m_pFactory = NULL;
        return FALSE;
    }
    // m_pFactory->AddRef(); // by Kim, Soomin
    return TRUE;
}

// Connecting to a chat server via ChatSock is a 2-step process.
// Since we don't know if the chat server is a MIC or IRC server,
// we call the socket factory and let it sort out the protocol issues.
// The socket factory will create a socket.
// Once we have the socket, we can then connect and log into the server.
BOOL CBaseSocket::FConnect(PEC_CONNINFO pcInfo, HWND hNotifyWnd)
{
    HRESULT hr = NOERROR;
    BOOL fDoneBak = FALSE; // have we tried the backup nick?
    BOOL fLoop = TRUE;

```

```

TCHAR          cBackupIDSuffix = '0';

ASSERT(pcInfo && m_pFactory);

if (FConnected())
    return TRUE;

if (hNotifyWnd)
    ::PostMessage(hNotifyWnd, CMD_CONNECT_CONNECTING, 0, 0);

m_bCanceled = FALSE;
TRACE("CBaseSocket: m_pFactory->HrMakeSocket(%s, %lx)\n", pcInfo->szServer, &m_pics);
hr = m_pFactory->HrMakeSocket(pcInfo->szServer, &m_pics);
if (FAILED(hr))
    goto LReturn;

if (m_pics->HrIsMicSocket() != NOERROR)
{
    ::MessageBox(NULL, "This is not an MIC socket!\n", "CBaseSocket",
    MB_OK);
    goto LReturn;
}

if (hNotifyWnd)
    ::PostMessage(hNotifyWnd, CMD_CONNECT_LOGIN, 0, 0);

CS_CONNINFO cInfo;
::ZeroMemory(&cInfo, sizeof(CS_CONNINFO));
cInfo.dwcb = sizeof(CS_CONNINFO);
cInfo.bType = pcInfo->fAuthenticate ? CS_CONNECT_AUTHENTICATE :
CS_CONNECT_ANONYMOUS;
cInfo.pvNick = (PVOID)pcInfo->szNick;
cInfo.pvUser = (PVOID)pcInfo->szUserName;
cInfo.pvPass = (PVOID)pcInfo->szPass;

// Loop till we have a result on the login
while (fLoop)
{
    TRACE0("CBaseSocket::FConnect: m_pics->HrLoginA(&cInfo);\n");
    hr = m_pics->HrLoginA(&cInfo);
    if (FAILED(hr))
    {
        goto LReturn;
    }

    // Now check the Wait Q for acknowledgement
    PCS_MSGBASE pcsMsg;
    TRACE0("CBaseSocket::FConnect: m_pics->HrWaitForMsg(&pcsMsg,
INFINITE);\n");
    hr = m_pics->HrWaitTillMsgType(CSMMSG_TYPE_LOGIN, &pcsMsg, pcInfo->dwTimeOut);
    TRACE("CBaseSocket::FConnect: m_pics->HrWaitForMsg returned
%lx\n", hr);

    if (FAILED(hr))
    {

```

```

// Did we log in or get an error?
// ASSERT(pcsMsg);
// switch (pcsMsg->csMsgType)
// {
// case CSMSG_TYPE_ERROR:
// {
//     TRACE0("CBaseSocket: CSMSG_TYPE_ERROR\n");
//     PCS_ERROR pErr = (PCS_ERROR)(pcsMsg + 1);
//     hr = pErr->hr;
//     if (hr == CS_E_ILLEGALUSER)
//     {
//         FOnSocketError(hr);
//         continue;
//     }
//     if ((hr == CS_E_ALIASINUSE) && !fDoneBak && pcInfo->szNickBak)
//     {
//         int n = lstrlen(pcInfo->szNickBak);
//         if (n > 1)
//         {
//             pcInfo->szNickBak[n-1] = cBackupIDSuffix;
//             cInfo.pvNick = (PVOID)pcInfo->szNickBak;
//             if (cBackupIDSuffix++ >= '9')
//             {
//                 fDoneBak = TRUE;
//             }
//             if (hNotifyWnd)
//             {
//                 ::PostMessage(hNotifyWnd,
//                     CMD_CONNECT_BACKUPID, 0, 0);
//             }
//             continue;
//         }
//     }
//     break;
// case CSMSG_TYPE_LOGIN:
//     TRACE0("CBaseSocket: CSMSG_TYPE_LOGIN\n");
//     hr = NOERROR;
//     break;
// default:
//     ASSERT(FALSE);
//     break;
// }
// goto LReturn;

FOnLogin(); // call the virtual function
fLoop = FALSE;
::HrFreeMsg(pcsMsg);
}

LReturn:
// If successful, save the socket
if (SUCCEEDED(hr))

```

```

    {
        TRACE0("CBaseSocket::FConnect: FStartMessageThread();\n");
        // Start a thread to read messages from the message que
        if (!FStartMessageThread())
        {
            FOnSocketError(E_OUTOFMEMORY);          // virtual function
call
            CleanUp();
            return FALSE;
        }
        return TRUE;
    }

    // Failure
    // On Cancel, m_pics->HrCloseSocket() may already been called by
    FCloseChatSocket().
    if (!m_bCanceled && m_pics)
    {
        TRACE0("CBaseSocket::FConnect: m_pics->HrCloseSocket();\n");
        m_pics->HrCloseSocket();
        TRACE0("CBaseSocket::FConnect: m_pics->Release();\n");
        m_pics->Release();
        m_pics = NULL;
    }

    // FOnSocketError(hr);          // virtual function call          // Doesn't work
    for OnCancel...
    // TRACE0("CBaseSocket::FConnect returned FALSE\n");
    return FALSE;
}

// Disconnect from the server.
// Cancels any in-progress connection attempts or other blocking operations.
BOOL CBaseSocket::FDisconnect()
{
    // As long as m_pFactory is alive, HrCancelMakeSocket() will have
    // the same effect as calling HrCloseSocket() on a ChatSocket.
    m_bCanceled = TRUE;
    FCloseChatSocket();
    if (m_pFactory)
    {
        TRACE0("CBaseSocket: m_pFactory->HrCancelMakeSocket();\n");
        HRESULT hr = m_pFactory->HrCancelMakeSocket();
        m_pFactory->Release();
        // m_pFactory = NULL;          // to prevent calling m_pFactory-
        >HrCancelMakeSocket() again
        if (FAILED(hr))
        {
            FOnSocketError(hr);          // virtual function call
            return FALSE;
        }
    }

    // m_pThread = NULL;
    // m_hThread = FALSE;
    return TRUE;
}

```

BaseSock.cpp

```
// Does this server support anonymous logins?
BOOL CBaseSocket::FCanAnonymous()
{
    // Grab security details
    PCS_SECURITY pcsSecurity;
    HRESULT hr = m_pics->HrGetSecurityInfo(&pcsSecurity);
    if (FAILED(hr))
    {
        return TRUE;        // assume anon
    }
    return pcsSecurity->fAnonAllowed;
}

// Uses m_pics->HrCreateChannelA() to create a channel; if a channel
// with the specified name already exists, this function simply joins it.
BOOL CBaseSocket::FCreateJoinChannel(PEC_CHANNELINFO pChanInfo)
{
    ASSERT(pChanInfo);
    ASSERT(m_pics);

    CS_CINFO cInfo;
    ::ZeroMemory(&cInfo, sizeof(CS_CINFO));    // This line was missing!!!!
    It's was the BUG! Nov 21thu'96
    cInfo.dwcb = sizeof(CS_CINFO);
    cInfo.dwType = pChanInfo->dwType;
    cInfo.dwFlags = pChanInfo->dwFlags;
    // if channel exists, join it, else create a new one
    cInfo.bCreateFlags = CS_CHANNEL_CREATE_JOIN;
    cInfo.pvChannelName = (PVOID)pChanInfo->szName;
    cInfo.pvTopic = (PVOID)pChanInfo->szTopic;
    cInfo.pvPassword = pChanInfo->szPass;
    cInfo.dwcUserMax = pChanInfo->cUsersMax;

    TRACE("CBaseSocket: m_pics->HrCreateChannelA(&cInfo) - %s\n",
(char*)cInfo.pvChannelName);
    HRESULT hr = m_pics->HrCreateChannelA(&cInfo);

    if (FAILED(hr))
    {
        FOnSocketError(hr);        // virtual function call
        return FALSE;
    }
    return TRUE;
}

BOOL CBaseSocket::FConnected()
{
    return (m_pics && (NOERROR == m_pics->HrIsConnected()));
}

BOOL CBaseSocket::ChangeNick(LPCSTR nick)
{
    ASSERT(m_pics);

    HRESULT hr = m_pics->HrChangeNickA((char*)nick);
    if (FAILED(hr))
    {

```

```

        FOnSocketError(hr);        // virtual function call
        return FALSE;
    }
    return TRUE;
}

// Waits for a message to arrive on the message queue.
// Calling FCloseChatSocket on the socket will cause this method to return
FALSE immediately.
// Dispatches the received message using the overrideable virtual methods
of CBaseSocket.
BOOL CBaseSocket::FWaitForMessage()
{
    if (!m_pics)
        return FALSE;
    m_pics->AddRef();                // increase the ref count so that
we can be sure that                // the socket object doesn't go
    TRACE0("m_pics->AddRef();\n");   away until this function ends...
    m_nPICSRef++;

    PCS_MSGBASE pcsMsg;
    while (SUCCEEDED(m_pics->HrWaitForMsg(&pcsMsg, INFINITE)))
    {
        DebugMessageType("=> CBaseSocket::FWaitForMessage", pcsMsg->
>csMsgType);
        switch (pcsMsg->csMsgType)    // BYTE
        {
            default:
                FUnknownMessage(pcsMsg);    // virtual function call
                break;

            case CSMSG_TYPE_ERROR:
                {
                    PCS_ERROR pErr = MSGBASE_TO_MSG(pcsMsg, PCS_ERROR);
                    FOnSocketError(pErr->hr);    // virtual function call
                }
                break;

            case CSMSG_TYPE_ADDCHANNEL:
                {
                    PCS_MSGCHANNEL pMsgCh = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGCHANNEL);
                    FOnAddChannel(pMsgCh);    // virtual function call;
                    Ref++
                }
                break;

            case CSMSG_TYPE_PRIVATEMSG:
                {
                    PCS_MSGPRIVATE pMsgPrivate = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGPRIVATE);
                    FOnPrivateMsg(pMsgPrivate);    // virtual function call
                }
                break;

            case CSMSG_TYPE_QUERYDATA:

```

```

        {
            PCS_PROPERTY      pcsProp = MSGBASE_TO_MSG(pcsMsg,
PCS_PROPERTY);
            FParseQueryData(pcsProp);      // virtual function call
        }
        break;

        case CSMSG_TYPE_INVITE:
        {
            PCS_MSGINVITE      pcsInvite = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGINVITE);
            FOnInvite(pcsInvite);          // virtual function call
        }
        break;
    }
    ::HrFreeMsg(pcsMsg);
}
if (m_pics)
{
    m_pics->Release();
    m_nPICSRef--;
    TRACE0("m_pics->Release();\n");      // the socket object doesn't
go away until this function ends...
    // m_pics = NULL;
}
TRACE0("End of CBaseSocket::FWaitForMessage()\n");
return TRUE;
}

// Performs default parsing on query data messages.
// If this method is not overridden, it will call FOnPropString and
FOnPropBuffer
// as appropriate for the query data.
BOOL CBaseSocket::FParseQueryData(PCS_PROPERTY pcsProp)
{
    return TRUE;
}

/*
ASSERT(pcsProp);

if (!pcsProp->picsProperty)
    return FALSE;

DWORD dwc;
if (FAILED((pcsProp->picsProperty)->HrGetCount(&dwc)))
    return FALSE;

BOOL fRet = TRUE;
for (DWORD dwi=1; dwi <= dwc && fRet; dwi++)
{
    CS_PROPDATA cspd;
    if (SUCCEEDED((pcsProp->picsProperty)->HrGetProperty(&cspd,
dwi)))
    {
        if (!cspd.pbData || !cspd.fAnsi) // For simplicity's sake,
ANSI only.
        {
            continue;

```



```

    }
    fRet = (cspd.fString)
        ? FOnPropString(cspd.csPropType, (CHAR*)cspd.pbData)
        : FOnPropBuffer(cspd.csPropType, cspd.pbData,
cspd.dwcb);
    }
    if (pcsProp->fLastRecord)
    {
        TRACE("This was the last record.\n");
        // What should I do?
    }
    return fRet;
*/
}

BOOL CBaseSocket::FStartMessageThread()
{
    WaitForMsgThread();

    m_pThread = AfxBeginThread(SocketThreadProc, (LPVOID)this);
    ASSERT(m_pThread);
    m_hThread = m_pThread->m_hThread; // Save the handle
    TRACE("CBaseSocket created a thread for SocketThreadProc [%lx]\n",
        m_pThread->m_nThreadID);
    return (m_pThread != NULL);
}

// static
UINT CBaseSocket::SocketThreadProc(LPVOID pvData)
{
    ASSERT(pvData);
    CBaseSocket* pbasesocket = (CBaseSocket*)pvData;
    return pbasesocket->FWaitForMessage();
}

////////////////////////////////////
////
// Operations via Chat Socket

BOOL CBaseSocket::FSendPrivAnsiText(char* szNickTo, char* szText)
{
    ASSERT(m_pics);

    CS_PRIVMSGINFO msg;
    msg.dwcb = sizeof(CS_PRIVMSGINFO);
    msg.dwUserID = 0; // 0 if not available
    msg.pvNickTo = szNickTo; // name of the user to send
this msg to
    msg.fData = FALSE; //
    msg.pbData = (BYTE*)szText; // data to send
    msg.dwcbData = 0; // 0 means null terminated
string
    HRESULT hr = m_pics->HrSendPrivMsgA(&msg);
    if (FAILED(hr))
    {
        FOnSocketError(hr); // virtual function call
    }
}

```

```

        return FALSE;
    }
    return TRUE;
}

BOOL CBaseSocket::FSendPrivData(char* szNickTo, BYTE* pbData, DWORD dwcb)
{
    ASSERT(m_pics);

    CS_PRIVMSGINFO    msg;
    msg.dwcb           = sizeof(CS_PRIVMSGINFO);
    msg.dwUserID       = 0;           // 0 if not available
    msg.pvNickTo       = szNickTo;   // name of the user to send this
msg to
    msg.fData          = TRUE;
    msg.pbData         = pbData;     // data to send
    msg.dwcbData       = dwcb;       // 0 means null terminated
string
    HRESULT hr = m_pics->HrSendPrivMsgA(&msg);
    if (FAILED(hr))
    {
        FOnSocketError(hr);         // virtual function call
        return FALSE;
    }
    return TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////
// Debugging

#ifdef _DEBUG
void DebugMessageType(LPCSTR t, CSMSG_TYPE c)
{
    switch (c)
    {
        case CSMSG_TYPE_NONE:
            TRACE1("%s\tCSMSG_TYPE_NONE\n", t);
            break;
            // socket, channel
        case CSMSG_TYPE_ERROR:
            TRACE1("%s\tCSMSG_TYPE_ERROR\n",
t); break;
            // socket
        case CSMSG_TYPE_LOGIN:
            TRACE1("%s\tCSMSG_TYPE_LOGIN\n",
t); break;
            // channel
        case CSMSG_TYPE_TEXT_A:
            TRACE1("%s\tCSMSG_TYPE_TEXT_A\n",
t); break;
        case CSMSG_TYPE_TEXT_W:
            TRACE1("%s\tCSMSG_TYPE_TEXT_W\n",
t); break;
        case CSMSG_TYPE_DATA:
            TRACE1("%s\tCSMSG_TYPE_DATA\n", t);
            break;
            // socket
        case CSMSG_TYPE_ADDCHANNEL:
            TRACE1("%s\tCSMSG_TYPE_ADDCHANNEL\n", t); break;
            // channel
        case CSMSG_TYPE_ADDMEMBER:
            TRACE1("%s\tCSMSG_TYPE_ADDMEMBER\n", t); break;
    }
}

```

```

        case CSMSG_TYPE_GOTMEMLIST:
            TRACE1("%s\tCSMSG_TYPE_GOTMEMLIST\n", t);          break;
        case CSMSG_TYPE_DELMEMBER:
            TRACE1("%s\tCSMSG_TYPE_DELMEMBER\n", t);          break;
        case CSMSG_TYPE_DELCHANNEL:
            TRACE1("%s\tCSMSG_TYPE_DELCHANNEL\n", t);          break;
        case CSMSG_TYPE_MODEMEMBER:
            TRACE1("%s\tCSMSG_TYPE_MODEMEMBER\n", t);          break;
        case CSMSG_TYPE_MODECHANNEL: TRACE1("%s\tCSMSG_TYPE_MODECHANNEL\n",
t);          break;
        case CSMSG_TYPE_WHISPERTEXT_A:
            TRACE1("%s\tCSMSG_TYPE_WHISPERTEXT_A\n", t);      break;
        case CSMSG_TYPE_WHISPERTEXT_W:
            TRACE1("%s\tCSMSG_TYPE_WHISPERTEXT_W\n", t);      break;
        case CSMSG_TYPE_WHISPERDATA: TRACE1("%s\tCSMSG_TYPE_WHISPERDATA\n",
t);          break;
        case CSMSG_TYPE_NEWTOPIC:    TRACE1("%s\tCSMSG_TYPE_NEWTOPIC\n",
t);          break;
        // socket, channel
        case CSMSG_TYPE_PROPERTYDATA: TRACE1("%s\tCSMSG_TYPE_PROPERTYDATA\n",
t);          break;
        // socket, channel
        case CSMSG_TYPE_QUERYDATA:    TRACE0(".");          break;
        //
        TRACE1("%s\tCSMSG_TYPE_QUERYDATA\n", t); break;
        // socket
        case CSMSG_TYPE_PRIVATEMSG:
            TRACE1("%s\tCSMSG_TYPE_PRIVATEMSG\n", t);          break;
        // channel
        case CSMSG_TYPE_NEWNICK:      TRACE1("%s\tCSMSG_TYPE_NEWNICK\n",
t);          break;
        // socket
        case CSMSG_TYPE_INVITE:      TRACE1("%s\tCSMSG_TYPE_INVITE\n",
t);          break;
        case CSMSG_TYPE_SERVERMSG_TEXT_A:
            TRACE1("%s\tCSMSG_TYPE_SERVERMSG_TEXT_A\n", t); break;
        case CSMSG_TYPE_SERVERMSG_TEXT_W:
            TRACE1("%s\tCSMSG_TYPE_SERVERMSG_TEXT_W\n", t); break;
        default:
            TRACE2("%s\tError! Beyond CSMSG_TYPE range(%d)!\n", t, c);
            break;
    }
}

DWORD PrintWin32Error(LPCSTR pszErrorString)
{
    LPVOID lpMsgBuf[FORMAT_MESSAGE_MAX_WIDTH_MASK + 1];
    // Retrieve the error code
    DWORD dwRc = ::GetLastError();
    // Search system message tables and resources for an error message
    // associated with the error returned
    if (::FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_MAX_WIDTH_MASK,
                        NULL,          // Message source
                        dwRc,          // Error Code
                        0,             // Language Specifier
                        (LPTSTR)lpMsgBuf, // Message buffer

```

BaseSock.cpp

```

                                FORMAT_MESSAGE_MAX_WIDTH_MASK,    //
Message Buffer Size            NULL))
                                TRACE("%s (%ld: %s)\n", pszErrorString, dwRc, lpMsgBuf);
                                return dwRc;
}
#endif                        // _DEBUG
```

BaseSock.h

```
//-----
//      CBaseSocket class
//
//      Copyright (C) 1996 Microsoft Corporation
//      All rights reserved.
//
//      Modification for MFC
//      (C) Programmed by Kim,
//      unichat
//      Information Technology Institue
//
//-----
#ifndef _BASESOCK_H
#define _BASESOCK_H

////////////////////////////////////
// class CBaseSocket

// Pointer to this structure is passed to CBaseSocket::HrConnect
typedef struct conninfo
{
    char* szServer;           // server to connect to
    char* szNick;             // Nickname to use
    char* szNickBak;          // A backup nick - optional
    char* szUserName;         // user name to use
    char* szPass;             // if szUserName is provided, so must
szPass
    BOOL fAuthenticate;       // should the connection be authenticated?
    DWORD dwTimeout;
} EC_CONNINFO, *PEC_CONNINFO;

// Pointer to this structure is passed to CBaseSocket::HrCreateJoinChannel
typedef struct channelinfo
{
    CHAR* szName;             // name of the channel to join
    CHAR* szTopic;            // if channel did not exist, and was therefore
                                // created for you, set the topic
to this value
    CHAR* szPass;             // password on this channel
    DWORD cUsersMax;          // how many users at max...0 if default
    DWORD dwType;             // channel type
    DWORD dwFlags;            // channel flags.
} EC_CHANNELINFO, *PEC_CHANNELINFO;

// CBaseSocket
// CBaseSocket is a wrapper around IChatSocket. It provides message-handling
code
// and wrappers around some IChatSocket functionality. If you need more
// functionality from ChatSock, use CBaseSocket::PSocket() to obtain a socket
// interface and then call ChatSock directly.
// Note: if you intend to keep that pointer around in other data structures,
// you should AddRef() it and then Release() it when that particular data
structure
// goes away. This will make your cleanup logic a lot more robust.

class CBaseSocket : public Object
{

```

BaseSock.h

```

public:
    BOOL ChangeNick(LPCSTR nick);
    CBaseSocket();
    virtual ~CBaseSocket();

    BOOL FInit();
    BOOL FConnect(PEC_CONNINFO pcInfo, HWND hNotifyWnd=NULL);
    BOOL FDisconnect();
    BOOL FCloseChatSocket();

    BOOL FCreateJoinChannel(PEC_CHANNELINFO pChanInfo);

    BOOL FCanAnonymous();
    BOOL FConnected();
    PICS PChatSocket();

    BOOL FSendPrivAnsiText(char* szNickTo, char* szText);
    BOOL FSendPrivData(char* szNickTo, BYTE* pbData, DWORD
dwcb);

    // Overrideables
    virtual BOOL FOnLogin() {
return TRUE; }
    virtual BOOL FOnSocketError(HRESULT hr) { return
TRUE; }
    virtual BOOL FOnAddChannel(PCS_MSGCHANNEL pMsg) { return TRUE; }
    virtual BOOL FOnPrivateMsg(PCS_MSGPRIVATE pMsg) { return TRUE; }
    virtual BOOL FOnInvite(PCS_MSGINVITE pcsInvite) { return TRUE; }
    // virtual BOOL FOnPropString(CSPROP_TYPE csType, char* sz) {
return TRUE; }
    // virtual BOOL FOnPropBuffer(CSPROP_TYPE csType, BYTE* pbBuffer,
DWORD dwcb) { return TRUE; }
    virtual BOOL FParseQueryData(PCS_PROPERTY pcsProp);
    // and if we got something we don't really handle..
    virtual BOOL FUnknownMessage(PCS_MSGBASE pMsg) { return TRUE; }

protected:
    static UINT SocketThreadProc(LPVOID pvData);
    void Cleanup();

    // BOOL FCloseChatSocket();
    void SetSocket(PICS pics);
    BOOL FStartMessageThread();
    BOOL FWaitForMessage();
    void WaitForMsgThread();

    PICS m_pics;
    PICS_FACTORY m_pFactory;
    CWinThread* m_pThread;
    HANDLE m_hThread; // Save the handle for
::WaitSingleObject
    BOOL m_bCanceled;
    int m_nPICSRef;
};

#ifdef _DEBUG
    void DebugMessageType(LPCSTR t, CSMSG_TYPE c);

```

BaseSock.h

```
        DWORD PrintWin32Error(LPCSTR pszErrorString);  
#else  
    #define DebugMessageType(t, c)          // Do nothing  
    #define PrintWin32Error(t)  
#endif  
  
#endif // _BASESOCK_H
```

Behavior.cpp

```
// Behavior.cpp: implementation of the CBehavior class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Behavior.h"

#include "Parser.h"
#include "ResMan.h"

#include "UC2Ani/DIB.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

extern CParser      gParser;
extern CResMan      gResMan;

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
CBehavior::CBehavior()
{
    m_nID = 0;
    m_nCells = 0;
    m_aCell = NULL;
    m_sDisp = CSize(0, 0);
}

CBehavior::~CBehavior()
{
    if (m_aCell)
        delete [] m_aCell;
}

// Loop in a line
BOOL CBehavior::Load(const int nID)
{
    // 0=1, (0,10,-1) (1,10,5,0,-1) (2,10,10,0,-1);      Walk
    // 1=1, (0) (1) (2); Changing direction
    // 2=1, (0) (1,10,0,0,-1) (2) (3) (4) (5) (6); Say
    m_nID = nID;
    m_nCells = gParser.CountOccurrencesUpto(')', ';');
    if (m_nCells <= 0)
    {
        return FALSE;
    }
    m_aCell = new Cell[m_nCells];
    gParser.GetValueRightToken(m_nRepeat, ',');
    for (int i=0; i < m_nCells; i++)
    {
        // initialize with default values
    }
}
```



```

m_aCell[i].wIO          = DEFAULT_IO;
m_aCell[i].nTicks = 10;
m_aCell[i].ptDisp = CPoint(0, 0);
m_aCell[i].nSI          = -1;
char* p = gParser.SetLeftToken('(');
if (!p)
{
    CString msg;
    msg.Format("Syntax Error in Behavior (ID=%d)\\"", nID);
    AfxMessageBox(msg);
    return FALSE;
}
while (!IsNum(*p))
{
    switch (*p)
    {
        case '#': m_aCell[i].wIO |= OPACITY_75;          break;
        case '*': m_aCell[i].wIO |= OPACITY_50;          break;
        case '/': m_aCell[i].wIO |= OPACITY_25;          break;
        case '|': m_aCell[i].wIO |= OPACITY_12;          break;
        case '~': m_aCell[i].wIO |= OPACITY_0;           break;
        case '-': m_aCell[i].wIO |= IMAGE_FLIP;          break;
        case '^': m_aCell[i].wIO |= NO_COLORKEY;         break;
    }
    gParser.SetCurrent(++p);
}
p = gParser.GetValueRightToken(m_aCell[i].nID, ')', ','); //
priority is for ')'
if (!p)
{
    TRACE0("Error reading in CBehavior::Load()\n");
    return FALSE;
}
if (*p == '(' || *p == ';') // next brace begins or end
of...
    continue;

p = gParser.GetValueRightToken(m_aCell[i].nTicks, ')', ',');
if (!p)
{
    m_aCell[i].nTicks = 5; // Although not found, the
parameter can be changed
    continue;
}
if (*p == '(' || *p == ';')
    continue;

p = gParser.GetValueRightToken(m_aCell[i].ptDisp.x, ')', ',');
if (!p)
{
    m_aCell[i].ptDisp.x = 0;
    continue;
}
else
    m_sDisp.cx += m_aCell[i].ptDisp.x;
if (*p == '(' || *p == ';')
    continue;

```

```

        p = gParser.GetValueRightToken(m_aCell[i].ptDisp.y, ')', ',');
        if (!p)
        {
            m_aCell[i].ptDisp.y = 0;
            continue;
        }
        else
            m_sDisp.cy += m_aCell[i].ptDisp.y;
        if (*p == '(' || *p == ';')
            continue;

        CString strSndName;
        p = gParser.GetValueRightToken(strSndName, ')');
        if (!p)
        {
            m_aCell[i].nSI = -1;
            continue;
        }
        else
        {
            m_aCell[i].nSI = gResMan.GetWaveID(strSndName);
        }
    }
    return TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
LPCSTR    BEHAVIOR_NAME[] =
{
    "STANDF",    // States
    "STANDB",
    "STANDINGF",
    "STANDINGB",
    "MORPHF",
    "MORPHB",
    "MORPHINGF",
    "MORPHINGB",
    "DOZEF",
    "DOZEB",
    "RES_STATE1",
    "RES_STATE2",
    "WALKF",    // Movements
    "WALKB",
    "UPF",
    "UPB",
    "DOWNF",
    "DOWNB",
    "MORPHWALKF",
    "MORPHWALKB",
    "RES_MOVE1",
    "RES_MOVE2",
    "CHAT",    // Gestures
    "ENTER",
    "EXIT",
    "SMILE",
    "MAD",

```

```

        "HELLO",
        "CRY",
        "SCRATCH",
        "PICK",
        "SPECIAL",
        "WIGGLEB",
        "PUNCHF",
        "PUNCHB",
        "BEATENF",
        "BEATENB",
        // "TURN180",
        // "TURN360",
        "RES_ACTION1",
        "RES_ACTION2"
    };

CActorDesc::CActorDesc()
{
    m_aBeh      = NULL;
    m_nBehs     = 0;
}

CActorDesc::~CActorDesc()
{
    if (m_aBeh)
        delete [] m_aBeh;
}

// If the requested Behavior ID is not found in this Actor Description,
// return NULL. CResMan::GetActorBehavior will deal with that case.
CBehavior* CActorDesc::GetBehavior(const int nID)
{
    for (int i=0; i < m_nBehs; i++)
    {
        if (m_aBeh[i].GetID() == nID)
            return &m_aBeh[i];
    }
    return NULL;
}

BOOL CActorDesc::Load(CTextFileBuffer& tfb)
{
    // #ACTOR=aman_000,°Å°I³²0,50;
    gParser.GetValueRightToken(m_strResName, ',');
    gParser.GetValueRightToken(m_strNick, ',');
    gParser.GetValueRightToken(m_nMSPT, ',', ',');

    if (m_nBehs <= 0)
        return TRUE;

    m_aBeh = new CBehavior[m_nBehs];    // CActorDesc::SetNumBehaviors
    // Read overloaded behaviors
    for (int nBeh=0; (nBeh < m_nBehs) && tfb.ReadString(); )
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
    }
}

```

```

        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        CString strBuf;
        if (gParser.GetValueRightToken(strBuf, '='))
        {
            int nID = GetIDFromString(strBuf);
            if (nID >= 0)
            {
                m_aBeh[nBeh++].Load(nID);
            }
            else
            {
                strBuf += ": Behavior not indexed!";
                AfxMessageBox(strBuf);
            }
        }
    }
    return TRUE;
}

int CActorDesc::GetIDFromString(const CString& strName) const
{
    for (int i=0; i < AB_RES_ACTION2; i++)
    {
        if (lstrcmpi(BEHAVIOR_NAME[i], strName) == 0)
            return i;
    }
    return -1;
}

```

Behavior.h

```
// Behavior.h: interface for the CBehavior class.
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      unichat
//=====

#if !defined(AFX_BEHAVIOR_H__C78B9065_A908_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_BEHAVIOR_H__C78B9065_A908_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

enum ACTOR_BEHAVIORS
{
    AB_STANDF=0,          // State
    AB_STANDB,
    AB_STANDINGF,
    AB_STANDINGB,
    AB_MORPHF,
    AB_MORPHB,
    AB_MORPHINGF,
    AB_MORPHINGB,
    AB_DOZEF,
    AB_DOZEB,
    AB_RES_STATE1,        // reserved
    AB_RES_STATE2,
    AB_WALKF,             // Moving
    AB_WALKB,
    AB_UPF,
    AB_UPB,
    AB_DOWNF,
    AB_DOWNB,
    AB_MORPHWALKF,
    AB_MORPHWALKB,
    AB_RES_MOVE1,
    AB_RES_MOVE2,
    AB_CHAT,              // Gestures
    AB_ENTER,
    AB_EXIT,
    AB_SMILE,
    AB_MAD,
    AB_HELLO,
    AB_CRY,
    AB_SCRATCH,
    AB_PICK,
    AB_SPECIAL,
    AB_WIGGLEB,
    AB_PUNCHF,
    AB_PUNCHB,
    AB_BEATENF,
    AB_BEATENB,
    // AB_TURN180,
    // AB_TURN360,
    AB_RES_ACTION1,
```

Behavior.h

```

        AB_RES_ACTION2
    };

    //////////////////////////////////////
class CBehavior // Actor Behavior Description
{
public:
    CBehavior();
    ~CBehavior();

    typedef struct
    {
        int          nID;          // ID for the cut (frame)
        WORD  wIO;          // IMAGE_OPTION
        int          nTicks;        // ticks for duration
        CPoint       ptDisp;        // displacement
        int          nSI;          // Sound Index
    } Cell;

    BOOL  Load(const int nID);      // Behavior ID
    int   GetID() const            { return m_nID; }
    Cell* GetCell(const int n)     { return (n < m_nCells && n >= 0) ?
&m_aCell[n] : NULL; }
    int   GetNumCells() const      { return m_nCells; }
    int   GetRepeat() const        { return (m_nRepeat > 0) ?
m_nRepeat : -m_nRepeat; }
    //  BOOL  GetPendulum() const    { return (m_nRepeat < 0); }
    CSize GetDisplacement() const  { return m_sDisp; }
    BOOL  IsNum(const char c) const { return (c >= '0' && c <= '9'); }

protected:
    int          m_nID;          // Behavior ID
    int          m_nCells;
    Cell* m_aCell;          // array of cells
    int          m_nRepeat;      // Repeat count, if (m_nRepeat < 0)
bPendulum should be set TRUE
    //  BOOL  m_bReturn;          // Return to Original position after
repetition.
    //  DWORD m_nRand;            // Random waiting range
    CSize m_sDisp;          // accumulated displacement
};

    //////////////////////////////////////
// Actor Description
class CTextFileBuffer;

class CActorDesc
{
public:
    CActorDesc();
    ~CActorDesc();

    BOOL  Load(CTextFileBuffer& tfb);
    void   SetNumBehaviors(const int n) { m_nBehs = n; }
    int   GetNumBehaviors() const      { return m_nBehs; }
    CString* GetNick()                  { return &m_strNick; }
};

```

Behavior.h

```
CString*   GetResName()           { return &m_strResName; }
}
int        GetMSPT() const        { return m_nMSPT; }
}
CBehavior* GetBehavior(const int nID);

protected:
    int        GetIDFromString(const CString& strName) const;

    CString    m_strNick;
    CString    m_strResName;
    int        m_nMSPT;           // Milliseconds per Tick
    int        m_nBehs;
    CBehavior* m_aBeh;
};

#endif //
!defined(AFX_BEHAVIOR_H__C78B9065_A908_11D1_80E2_080009B9F339__INCLUDED_)
```

```

//
// Blocksock.cpp
// (CBlockingSocketException, CBlockingSocket, CHttpBlockingSocket)
//
// Copyright Quartermview 1999
// joseph j. kim (jokim@quartermview.com)
//

#include <stdafx.h>
#include "blocksock.h"

// Class CBlockingSocketException
IMPLEMENT_DYNAMIC(CBlockingSocketException, CException)

CBlockingSocketException::CBlockingSocketException(char* pchMessage)
{
    m_strMessage = pchMessage;
    m_nError = WSAGetLastError();
}

BOOL CBlockingSocketException::GetErrorMessage(LPTSTR lpstrError, UINT
nMaxError,
        PUINT pnHelpContext /*= NULL*/)
{
    char text[200];
    if(m_nError == 0) {
        wsprintf(text, "%s error", (const char*) m_strMessage);
    }
    else {
        wsprintf(text, "%s error #%d", (const char*) m_strMessage,
m_nError);
    }
    strncpy(lpstrError, text, nMaxError - 1);
    return TRUE;
}

// Class CBlockingSocket
IMPLEMENT_DYNAMIC(CBlockingSocket, CObject)

void CBlockingSocket::Cleanup()
{
    // doesn't throw an exception because it's called in a catch block
    if(m_hSocket == NULL) return;
    VERIFY(closesocket(m_hSocket) != SOCKET_ERROR);
    m_hSocket = NULL;
}

void CBlockingSocket::Create(int nType /* = SOCK_STREAM */)
{
    ASSERT(m_hSocket == NULL);
    if((m_hSocket = socket(AF_INET, nType, 0)) == INVALID_SOCKET) {
        throw new CBlockingSocketException("Create");
    }
}

```



```

void CBlockingSocket::Bind(LPCSOCKADDR psa)
{
    ASSERT(m_hSocket != NULL);
    if(bind(m_hSocket, psa, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        throw new CBlockingSocketException("Bind");
    }
}

void CBlockingSocket::Listen()
{
    ASSERT(m_hSocket != NULL);
    if(listen(m_hSocket, 5) == SOCKET_ERROR) {
        throw new CBlockingSocketException("Listen");
    }
}

BOOL CBlockingSocket::Accept(CBlockingSocket& sConnect, LPSOCKADDR psa)
{
    ASSERT(m_hSocket != NULL);
    ASSERT(sConnect.m_hSocket == NULL);
    int nLengthAddr = sizeof(SOCKADDR);
    sConnect.m_hSocket = accept(m_hSocket, psa, &nLengthAddr);
    if(sConnect == INVALID_SOCKET) {
        // no exception if the listen was canceled
        if(WSAGetLastError() != WSAEINTR) {
            throw new CBlockingSocketException("Accept");
        }
        return FALSE;
    }
    return TRUE;
}

void CBlockingSocket::Close()
{
    if (NULL == m_hSocket)
        return;

    if(closesocket(m_hSocket) == SOCKET_ERROR) {
        // should be OK to close if closed already
        throw new CBlockingSocketException("Close");
    }
    m_hSocket = NULL;
}

void CBlockingSocket::Connect(LPCSOCKADDR psa)
{
    ASSERT(m_hSocket != NULL);
    // should timeout by itself
    if(connect(m_hSocket, psa, sizeof(SOCKADDR)) == SOCKET_ERROR) {
        throw new CBlockingSocketException("Connect");
    }
}

int CBlockingSocket::Write(const char* pch, const int nSize, const int nSecs)
{
    int nBytesSent = 0;

```

```

    int nBytesThisTime;
    const char* pch1 = pch;
    do {
        nBytesThisTime = Send(pch1, nSize - nBytesSent, nSecs);
        nBytesSent += nBytesThisTime;
        pch1 += nBytesThisTime;
    } while(nBytesSent < nSize);
    return nBytesSent;
}

int CBlockingSocket::Send(const char* pch, const int nSize, const int nSecs)
{
    ASSERT(m_hSocket != NULL);
    // returned value will be less than nSize if client cancels the reading
    FD_SET fd = {1, m_hSocket};
    TIMEVAL tv = {nSecs, 0};
    if(select(0, NULL, &fd, NULL, &tv) == 0) {
        throw new CBlockingSocketException("Send timeout");
    }
    int nBytesSent;
    if((nBytesSent = send(m_hSocket, pch, nSize, 0)) == SOCKET_ERROR) {
        throw new CBlockingSocketException("Send");
    }
    return nBytesSent;
}

int CBlockingSocket::Receive(char* pch, const int nSize, const int nSecs)
{
    ASSERT(m_hSocket != NULL);
    FD_SET fd = {1, m_hSocket};
    TIMEVAL tv = {nSecs, 0};
    if(select(0, &fd, NULL, NULL, &tv) == 0) {
        throw new CBlockingSocketException("Receive timeout");
    }
    int nBytesReceived;
    if((nBytesReceived = recv(m_hSocket, pch, nSize, 0)) == SOCKET_ERROR) {
        throw new CBlockingSocketException("Receive");
    }
    return nBytesReceived;
}

int CBlockingSocket::ReceiveDatagram(char* pch, const int nSize, LPSOCKADDR
psa, const int nSecs)
{
    ASSERT(m_hSocket != NULL);
    FD_SET fd = {1, m_hSocket};
    TIMEVAL tv = {nSecs, 0};
    if(select(0, &fd, NULL, NULL, &tv) == 0) {
        throw new CBlockingSocketException("Receive timeout");
    }

    // input buffer should be big enough for the entire datagram
    int nFromSize = sizeof(SOCKADDR);
    int nBytesReceived = recvfrom(m_hSocket, pch, nSize, 0, psa,
&nFromSize);
    if(nBytesReceived == SOCKET_ERROR) {

```

```

        throw new CBlockingSocketException("ReceiveDatagram");
    }
    return nBytesReceived;
}

int CBlockingSocket::SendDatagram(const char* pch, const int nSize,
LPCSOCKADDR psa, const int nSecs)
{
    ASSERT(m_hSocket != NULL);
    FD_SET fd = {1, m_hSocket};
    TIMEVAL tv = {nSecs, 0};
    if(select(0, NULL, &fd, NULL, &tv) == 0) {
        throw new CBlockingSocketException("Send timeout");
    }

    int nBytesSent = sendto(m_hSocket, pch, nSize, 0, psa,
sizeof(SOCKADDR));
    if(nBytesSent == SOCKET_ERROR) {
        throw new CBlockingSocketException("SendDatagram");
    }
    return nBytesSent;
}

void CBlockingSocket::GetPeerAddr(LPSOCKADDR psa)
{
    ASSERT(m_hSocket != NULL);
    // gets the address of the socket at the other end
    int nLengthAddr = sizeof(SOCKADDR);
    if(getpeername(m_hSocket, psa, &nLengthAddr) == SOCKET_ERROR) {
        throw new CBlockingSocketException("GetPeerName");
    }
}

void CBlockingSocket::GetSockAddr(LPSOCKADDR psa)
{
    ASSERT(m_hSocket != NULL);
    // gets the address of the socket at this end
    int nLengthAddr = sizeof(SOCKADDR);
    if(getsockname(m_hSocket, psa, &nLengthAddr) == SOCKET_ERROR) {
        throw new CBlockingSocketException("GetSockName");
    }
}

//static
CSockAddr CBlockingSocket::GetHostByName(const char* pchName, const USHORT
ushPort /* = 0 */)
{
    hostent* pHostEnt = gethostbyname(pchName);
    if(pHostEnt == NULL) {
        throw new CBlockingSocketException("GetHostByName");
    }
    ULONG* pulAddr = (ULONG*) pHostEnt->h_addr_list[0];
    SOCKADDR_IN sockTemp;
    sockTemp.sin_family = AF_INET;
    sockTemp.sin_port = htons(ushPort);
    sockTemp.sin_addr.s_addr = *pulAddr; // address is already in network
byte order

```

Blocksock.cpp

```

        return sockTemp;
    }

//static
const char* CBlockingSocket::GetHostByAddr(LPCSOCKADDR psa)
{
    hostent* pHostEnt = gethostbyaddr((char*) &((LPSOCKADDR_IN) psa)
        ->sin_addr.s_addr, 4, PF_INET);
    if(pHostEnt == NULL) {
        throw new CBlockingSocketException("GetHostByAddr");
    }
    return pHostEnt->h_name; // caller shouldn't delete this memory
}

// Class CHttpBlockingSocket
IMPLEMENT_DYNAMIC(CHttpBlockingSocket, CBlockingSocket)

CHttpBlockingSocket::CHttpBlockingSocket()
{
    m_pReadBuf = new char[nSizeRecv];
    m_nReadBuf = 0;
}

CHttpBlockingSocket::~CHttpBlockingSocket()
{
    delete [] m_pReadBuf;
}

// reads an entire header line through CRLF (or socket close)
// inserts zero string terminator, object maintains a buffer
int CHttpBlockingSocket::ReadHttpHeaderLine(char* pch, const int nSize, const
int nSecs)
{
    TRACE("\nCHttpBlockingSocket:ReadHttpHeaderLine");
    TRACE("\nargs: size-%d, nSecs-%d, pch-%s", nSize, nSecs, pch);

    int nBytesThisTime = m_nReadBuf;

    TRACE("    nBytesThisTime: %d", nBytesThisTime);

    int nLineLength = 0;
    char* pch1 = m_pReadBuf;
    char* pch2;
    do {
        // look for lf (assume preceded by cr)
        if((pch2 = (char*) memchr(pch1, '\n', nBytesThisTime)) != NULL)
        {
            ASSERT((pch2) > m_pReadBuf);
            ASSERT(*(pch2 - 1) == '\r');
            nLineLength = (pch2 - m_pReadBuf) + 1;
            if(nLineLength >= nSize) nLineLength = nSize - 1;
            memcpy(pch, m_pReadBuf, nLineLength); // copy the line to
caller

```

```

        m_nReadBuf -= nLineLength;
        memmove(m_pReadBuf, pch2 + 1, m_nReadBuf); // shift
remaining characters left
        break;
    }
    pch1 += nBytesThisTime;

    nBytesThisTime = Receive(m_pReadBuf + m_nReadBuf, nSizeRecv -
m_nReadBuf, nSecs);
    if(nBytesThisTime <= 0) { // sender closed socket or line longer
than buffer
        throw new CBlockingSocketException("ReadHeaderLine");
    }
    m_nReadBuf += nBytesThisTime;
}
while(TRUE);
*(pch + nLineLength) = '\0';
return nLineLength;
}

```

```

// reads remainder of a transmission through buffer full or socket close
// (assume headers have been read already)
int CHttpBlockingSocket::ReadHttpResponse(char* pch, const int nSize, const
int nSecs)
{

```

```

    int nBytesToRead, nBytesThisTime, nBytesRead = 0;
    if(m_nReadBuf > 0) { // copy anything already in the recv buffer
        memcpy(pch, m_pReadBuf, m_nReadBuf);
        pch += m_nReadBuf;
        nBytesRead = m_nReadBuf;
        m_nReadBuf = 0;
    }
    do { // now pass the rest of the data directly to the caller
        nBytesToRead = min(nSizeRecv, nSize - nBytesRead);
        nBytesThisTime = Receive(pch, nBytesToRead, nSecs);
        if(nBytesThisTime <= 0) break; // sender closed the socket
        pch += nBytesThisTime;
        nBytesRead += nBytesThisTime;
    }
    while(nBytesRead <= nSize);
    return nBytesRead;
}

```

```

void LogBlockingSocketException(LPVOID pParam, char* pch,
CBlockingSocketException* pe)
{
    // pParam holds the HWND for the destination window (in another thread)
    CString strGmt = CTime::GetCurrentTime().FormatGmt("%m/%d/%y %H:%M:%S
GMT");
    char text1[200], text2[50];
    pe->GetErrorMessage(text2, 49);
    wsprintf(text1, "WINSOCK ERROR--%s %s -- %s\r\n", pch, text2, (const
char*) strGmt);
    ::SendMessage((HWND) pParam, EM_SETSEL, (WPARAM) 65534, 65535);
    ::SendMessage((HWND) pParam, EM_REPLACESEL, (WPARAM) 0, (LPARAM)
text1);
}

```

```

// blocksock.h

// needs winsock.h in the precompiled headers

#ifndef __BLOCKSOCK_H
#define __BLOCKSOCK_H

typedef const struct sockaddr* LPCSOCKADDR;

class CBlockingSocketException : public CException
{
    DECLARE_DYNAMIC(CBlockingSocketException)
public:
    // Constructor
    CBlockingSocketException(char* pchMessage);

public:
    ~CBlockingSocketException() {}
    virtual BOOL GetErrorMessage(LPTSTR lpstrError, UINT nMaxError,
        PUINT pnHelpContext = NULL);
private:
    int m_nError;
    CString m_strMessage;
};

extern void LogBlockingSocketException(LPVOID pParam, char* pch,
    CBlockingSocketException* pe);

class CSockAddr : public sockaddr_in {
public:
    // constructors
    CSockAddr()
    { sin_family = AF_INET;
      sin_port = 0;
      sin_addr.s_addr = 0; } // Default
    CSockAddr(const SOCKADDR& sa) { memcpy(this, &sa, sizeof(SOCKADDR)); }
    CSockAddr(const SOCKADDR_IN& sin) { memcpy(this, &sin,
sizeof(SOCKADDR_IN)); }
    CSockAddr(const ULONG ulAddr, const USHORT ushPort = 0) // parms are
host byte ordered
    { sin_family = AF_INET;
      sin_port = htons(ushPort);
      sin_addr.s_addr = htonl(ulAddr); }
    CSockAddr(const char* pchIP, const USHORT ushPort = 0) // dotted IP
addr string
    { sin_family = AF_INET;
      sin_port = htons(ushPort);
      sin_addr.s_addr = inet_addr(pchIP); } // already network byte
ordered
    // Return the address in dotted-decimal format
    CString DottedDecimal()

```

```

        { return inet_ntoa(sin_addr); } // constructs a new CString
object
    // Get port and address (even though they're public)
    USHORT Port() const
    { return ntohs(sin_port); }
    ULONG IPAddr() const
    { return ntohl(sin_addr.s_addr); }
    // operators added for efficiency
    const CSockAddr& operator=(const SOCKADDR& sa)
    { memcpy(this, &sa, sizeof(SOCKADDR));
      return *this; }
    const CSockAddr& operator=(const SOCKADDR_IN& sin)
    { memcpy(this, &sin, sizeof(SOCKADDR_IN));
      return *this; }
    operator SOCKADDR()
    { return *((LPSOCKADDR) this); }
    operator LPSOCKADDR()
    { return (LPSOCKADDR) this; }
    operator LPSOCKADDR_IN()
    { return (LPSOCKADDR_IN) this; }
};

// member functions truly block and must not be used in UI threads
// use this class as an alternative to the MFC CSocket class
class CBlockingSocket : public CObject
{
    DECLARE_DYNAMIC(CBlockingSocket)
public:
    SOCKET m_hSocket;
    CBlockingSocket() { m_hSocket = NULL; }
    void Cleanup();
    void Create(int nType = SOCK_STREAM);
    void Close();
    void Bind(LPSOCKADDR psa);
    void Listen();
    void Connect(LPSOCKADDR psa);
    BOOL Accept(CBlockingSocket& s, LPSOCKADDR psa);
    int Send(const char* pch, const int nSize, const int nSecs);
    int Write(const char* pch, const int nSize, const int nSecs);
    int Receive(char* pch, const int nSize, const int nSecs);
    int SendDatagram(const char* pch, const int nSize, LPSOCKADDR psa,
        const int nSecs);
    int ReceiveDatagram(char* pch, const int nSize, LPSOCKADDR psa,
        const int nSecs);
    void GetPeerAddr(LPSOCKADDR psa);
    void GetSockAddr(LPSOCKADDR psa);
    static CSockAddr GetHostByName(const char* pchName,
        const USHORT ushPort = 0);
    static const char* GetHostByAddr(LPSOCKADDR psa);
    operator SOCKET()
    { return m_hSocket; }
};

class CHttpBlockingSocket : public CBlockingSocket
{
public:
    DECLARE_DYNAMIC(CHttpBlockingSocket)

```

Blocksock.h

```
enum {nSizeRecv = 1000}; // max receive buffer size (> hdr line length)
CHttpBlockingSocket();
~CHttpBlockingSocket();
int ReadHttpHeaderLine(char* pch, const int nSize, const int nSecs);
int ReadHttpResponse(char* pch, const int nSize, const int nSecs);
private:
    char* m_pReadBuf; // read buffer
    int m_nReadBuf; // number of bytes in the read buffer
};

#endif // __BLOCKSOCK_H
```


BMC.CPP

```
// BMC.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "BMC.h"

#include "MainFrm.h"
#include "BMCDoc.h"
#include "BMCView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CBMCApp

BEGIN_MESSAGE_MAP(CBMCApp, CWinApp)
    //{AFX_MSG_MAP(CBMCApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // NOTE - the ClassWizard will add and remove mapping macros
here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //{AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

////////////////////////////////////
// CBMCApp construction

CBMCApp::CBMCApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

////////////////////////////////////
// The one and only CBMCApp object

CBMCApp theApp;

////////////////////////////////////
// CBMCApp initialization

BOOL CBMCApp::InitInstance()
{
    if (!AfxSocketInit())
    {
        AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
        return FALSE;
    }
}
```

```

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();          // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic();    // Call this when linking to MFC
statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings();    // Load standard INI file options (including
MRU)

    // Register the application's document templates.  Document templates
    // serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CBMCDoc),
        RUNTIME_CLASS(CMainFrame),           // main SDI frame window
        RUNTIME_CLASS(CBMCView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

```

```

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
    // No message handlers
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CBMCApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CBMCApp commands

```

BMC.H

```
// BMC.h : main header file for the BMC application
//

#if !defined(AFX_BMC_H__C193E4A8_88DD_11D1_ACEA_080009B9F339__INCLUDED_)
#define AFX_BMC_H__C193E4A8_88DD_11D1_ACEA_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifdef __AFXWIN_H
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CBMCApp:
// See BMC.cpp for the implementation of this class
//

class CBMCApp : public CWinApp
{
public:
    CBMCApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CBMCApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CBMCApp)
    afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //      DO NOT EDIT what you see in these blocks of generated code
    !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#define AFX_BMC_H__C193E4A8_88DD_11D1_ACEA_080009B9F339__INCLUDED_
```

BMCDoc.cpp

```
// BMCDoc.cpp : implementation of the CBMCDoc class
//

#include "stdafx.h"
#include "BMC.h"

#include "BMCDoc.h"
#include "BMCView.h"

#include "../UC2Ani/DIB.h"
#include "../UC2Ani/PhSprite.h"

#include "PaletteDlg.h"
#include "InputDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CBMCDoc

IMPLEMENT_DYNCREATE(CBMCDoc, CDocument)

BEGIN_MESSAGE_MAP(CBMCDoc, CDocument)
    //{{AFX_MSG_MAP(CBMCDoc)
    ON_COMMAND(ID_FILE_LOADSPRITE, OnFileLoadsprite)
    ON_COMMAND(ID_FILE_LOADBKGND, OnFileLoadbkgnd)
    ON_COMMAND(ID_VIEW_PALETTE, OnViewPalette)
    ON_COMMAND(ID_VIEW_OUTLINE, OnViewOutline)
    ON_COMMAND(ID_FILE_SAVE_BKG, OnFileSaveBkg)
    ON_COMMAND(ID_FILE_SAVE_SPRITE, OnFileSaveSprite)
    ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_BKG, OnUpdateFileSaveBkg)
    ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_SPRITE, OnUpdateFileSaveSprite)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CBMCDoc construction/destruction

CBMCDoc::CBMCDoc()
{
    m_pBkgndDIB = NULL;
}

CBMCDoc::~CBMCDoc()
{
    if (m_pBkgndDIB)
        delete m_pBkgndDIB;
    m_SpriteList.RemoveAll();
}

BOOL CBMCDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())

```

```

        return FALSE;

// Tell the sprite list notification object where the view is
// so when we add sprites the dirty list can be maintained automatically.
CBMCView* pView = GetBMCView();
ASSERT(pView);
m_SpriteList.m_NotifyObj.SetView(pView);

// Create a new default background DIB just so
// there will be something to look at
CDIB* pDIB = new CDIB;
pDIB->Create(320, 240);
SetBackground(pDIB);
SetModifiedFlag(FALSE);

    return TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CBMCDoc serialization

void CBMCDoc::Serialize(CArchive& ar)
{
    CDocument::Serialize(ar);
    if (ar.IsStoring())
    {
        if (m_pBkgndDIB)
        {
            ar << (DWORD) 1; // say we have a bkgnd
            m_pBkgndDIB->Serialize(ar);
        }
        else
        {
            ar << (DWORD) 0; // say we have no bkgnd
        }
        m_SpriteList.Serialize(ar);
    }
    else
    {
        DWORD dw;
        // see if we have a background to load
        ar >> dw;
        if (dw != 0)
        {
            CDIB* pDIB = new CDIB;
            pDIB->Serialize(ar);
            // Attach it to the document
            SetBackground(pDIB);
        }
        // read the sprite list
        CBMCView* pView = GetBMCView();
        ASSERT(pView);
        m_SpriteList.m_NotifyObj.SetView(pView);
        m_SpriteList.Serialize(ar);

        SetModifiedFlag(FALSE);
        UpdateAllViews(NULL, 0, NULL);
    }
}

```

```

    }
}

/////////////////////////////////////////////////////////////////
// CBMCDoc diagnostics

#ifdef _DEBUG
void CBMCDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CBMCDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////
// CBMCDoc commands

void CBMCDoc::SetTitle(LPCTSTR lpszTitle)
{
    //    UNREFERENCED_PARAMETER(lpszTitle);

    //    CString strTitle("UniChat 2.0");
    //    if (lpszTitle)
    //        strTitle += lpszTitle;
    //    CDocument::SetTitle(strTitle);
    //    CDocument::SetTitle(lpszTitle);
}

/////////////////////////////////////////////////////////////////
// return a pointer to the off-screen buffered view
CBMCView* CBMCDoc::GetBMCView()
{
    POSITION pos = GetFirstViewPosition();
    ASSERT(pos);
    CBMCView* pView = (CBMCView*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CBMCView)));
    return pView;
}

void CBMCDoc::OnFileLoadbkgn()
{
    // Create a DIB to hold the image
    CDIB* pDIB = new CDIB;
    // Show the file open dialog for a DIB
    CInputDlg dlg;
    dlg.m_strData = "DataSource|file";
    if (dlg.DoModal() != IDOK)
        return;

    if (!pDIB->Load(dlg.m_strData, "c:\\uc2\\bmc\\master.pal"))
    {
        delete pDIB;
    }
}

```

```

        return;
    }
/*
    if (!pDIB->Load((char*)NULL, "master.pal"))
    {
        delete pDIB;
        return;
    }
*/
    CString* pstrFile = pDIB->GetName();
    SetTitle(*pstrFile);
    // Make sure this is an 8 bpp DIB
    BITMAPINFO* pBMI = pDIB->GetBitmapInfoAddress();
    ASSERT(pBMI);
    if (pBMI->bmiHeader.biBitCount != 8)
    {
        AfxMessageBox("Only 8 bpp DIBs are supported");
        delete pDIB;
        return;
    }
    // replace any existing background DIB with the new one
    if (!SetBackground(pDIB))
        delete pDIB;
}

void CBMCDoc::OnFileLoadsprite()
{
    // Create a sprite to hold the image
    CPhasedSprite* pSprite = new CPhasedSprite;
    // Show the file open dialog for a Sprite
    if (!pSprite->Load((char*)NULL, "master.pal"))
    {
        delete pSprite;
        return;
    }
    CString* pstrFileBkg = m_pBkgndDIB->GetName();
    CString* pstrFileSprite = pSprite->GetDIB()->GetName();
    CString strTitle;
    strTitle.Format("(B)%s, (S)%s", *pstrFileBkg, *pstrFileSprite);
    SetTitle(strTitle);
    // Make sure this is an 8 bpp DIB
    BITMAPINFO* pBMI = pSprite->GetDIB()->GetBitmapInfoAddress();
    ASSERT(pBMI);
    if (pBMI->bmiHeader.biBitCount != 8)
    {
        AfxMessageBox("Only 8 bpp DIBs are supported");
        delete pSprite;
        return;
    }
    // add it to the sprite list
    m_SpriteList.Insert(pSprite);
    // SetModifiedFlag(TRUE);

    // Tell the view about the new sprite
    GetBMCView()->NewSprite(pSprite);
}

// Set a new background DIB

```



```

BOOL CBMCDoc::SetBackground(CDIB* pDIB)
{
    // Delete any existing sprites
    m_SpriteList.RemoveAll();

    // Delete any existing background DIB and set the new one
    if (m_pBkgndDIB)
        delete m_pBkgndDIB;
    m_pBkgndDIB = pDIB;

    // Note that the doc has changed
    // SetModifiedFlag(TRUE);

    // Tell the view that it needs to create a new buffer and palette
    CBMCView* pView = GetBMCView();
    ASSERT(pView);
    return pView->NewBackground(m_pBkgndDIB);
}

void CBMCDoc::GetSceneRect(CRect& prc)
{
    if (!m_pBkgndDIB)
        return;
    m_pBkgndDIB->GetRect(prc);
}

void CBMCDoc::OnViewPalette()
{
    CPaletteDlg palDlg;

    CBMCView* pView = GetBMCView();
    if (pView)
        palDlg.SetPalette(pView->GetOSBPalette());
    palDlg.DoModal();
}

void CBMCDoc::OnViewOutline()
{
    CBMCView* pView = GetBMCView();
    pView->ToggleOutline();
}

void CBMCDoc::OnFileSaveBkg()
{
    if (m_pBkgndDIB)
    {
        CString* pstrFile = m_pBkgndDIB->GetName();
        // Reload to restore original bitmaps
        CDIB* pDIB = new CDIB;
        // Show the file open dialog for a DIB
        if (!pDIB->Load(*pstrFile))
        {
            delete pDIB;
            return;
        }
        pDIB->Save();
        delete pDIB;
    }
}

```

```

    }
}

void CBMCDoc::OnFileSaveSprite()
{
    CPhasedSprite* pS = GetBMCView()->GetCapturedSprite();
    if (pS)
    {
        CString* pstrFile = pS->GetDIB()->GetName();
        // Reload to restore original bitmaps
        CDIB* pDIB = new CDIB;
        // Show the file open dialog for a DIB
        if (!pDIB->Load(*pstrFile))
        {
            delete pDIB;
            return;
        }
        pDIB->Save();
        delete pDIB;
    }
}

void CBMCDoc::OnUpdateFileSaveBkg(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_pBkgndDIB != NULL);
}

void CBMCDoc::OnUpdateFileSaveSprite(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(GetBMCView()->GetCapturedSprite() != NULL);
}

```

BMCDoc.h

```
// BMCDoc.h : interface of the CBMCDoc class
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_BMCDOC_H__C193E4AE_88DD_11D1_ACEA_080009B9F339__INCLUDED_)
#define AFX_BMCDOC_H__C193E4AE_88DD_11D1_ACEA_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "../UC2Ani/SpritLst.h"

class CDIB;
class CBMCView;

class CBMCDoc : public CDocument
{
protected: // create from serialization only
    CBMCDoc();
    DECLARE_DYNCREATE(CBMCDoc)

// Attributes
public:
    CDIB*                GetBackground()    { return m_pBkgndDIB; }
    CSpriteList* GetSpriteList()    { return &m_SpriteList; }
    void                GetSceneRect(CRect& prc);

// Operations
public:
    BOOL                SetBackground(CDIB* pDIB);

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CBMCDoc)
    public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        virtual void SetTitle(LPCTSTR lpszTitle);
    //}AFX_VIRTUAL

// Implementation
public:
    virtual ~CBMCDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

private:
    CDIB*                m_pBkgndDIB;        // ptr to background DIB
    CSpriteList          m_SpriteList;      // sprite list

    CBMCView*            GetBMCView();

```

```
// Generated message map functions
protected:
```

```
    //{AFX_MSG(CBMCDoc)
    afx_msg void OnFileLoadsprite();
    afx_msg void OnFileLoadbkgnd();
    afx_msg void OnViewPalette();
    afx_msg void OnViewOutline();
    afx_msg void OnFileSaveBkg();
    afx_msg void OnFileSaveSprite();
    afx_msg void OnUpdateFileSaveBkg(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFileSaveSprite(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

```
};
```

```
////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
```

```
#endif //
```

```
!defined(AFX_BMCDOC_H__C193E4AE_88DD_11D1_ACEA_080009B9F339__INCLUDED_)
```

BMCView.cpp

```
// BMCView.cpp : implementation of the CBMCView class
//

#include "stdafx.h"
#include "BMC.h"

#include "BMCDoc.h"
#include "BMCView.h"
#include "MainFrm.h"

#include "../UC2Ani/DIB.h"
#include "../UC2Ani/DIBPal.h"
#include "../UC2Ani/PhSprite.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CBMCView

IMPLEMENT_DYNCREATE(CBMCView, COSBView)

BEGIN_MESSAGE_MAP(CBMCView, COSBView)
    //{AFX_MSG_MAP(CBMCView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
    ON_COMMAND(ID_VIEW_640x480, OnVIEW640x480)
    ON_COMMAND(ID_VIEW_800x600, OnVIEW800x600)
    //}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////
// CBMCView construction/destruction

CBMCView::CBMCView()
{
    m_bMouseCaptured = FALSE;
    m_pCapturedSprite = NULL;
    m_aSubPE = new PALETTEENTRY[256];
/*
    m_nPEs = 10;
    m_apPE = new LPPALETTEENTRY[m_nPEs];
    for (int i=0; i < m_nPEs; i++)
    {
        m_apPE[i] = new PALETTEENTRY[256];
    }
*/
}
}
```

BMCView.cpp

```

CBMCView::~CBMCView()
{
    if (m_aSubPE)
        delete [] m_aSubPE;
/*
    if (m_apPE)
    {
        for (int i=0; i < m_nPEs; i++)
        {
            if (m_apPE[i])
                delete [] m_apPE[i];
            m_apPE[i] = NULL;
        }
        delete [] m_apPE;
        m_apPE = NULL;
    }
*/
}

BOOL CBMCView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

////////////////////////////////////
// CBMCView drawing

void CBMCView::OnDraw(CDC* pDC)
{
    // CBMCDoc* pDoc = GetDocument();
    // ASSERT_VALID(pDoc);

    COSBView::OnDraw(pDC);
}

////////////////////////////////////
// CBMCView printing

BOOL CBMCView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CBMCView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CBMCView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CBMCView diagnostics

#ifdef _DEBUG
void CBMCView::AssertValid() const
{
    CView::AssertValid();
}

void CBMCView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CBMCDoc* CBMCView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CBMCDoc)));
    return (CBMCDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CBMCView message handlers

void CBMCView::OnInitialUpdate()
{
    COSBView::OnInitialUpdate();
}

// Create a new buffer and palette to match a new background DIB
BOOL CBMCView::NewBackground(CDIB* pDIB)
{
    m_pCapturedSprite = NULL;

    /*
    CDIBPal pal;
    pal.Load((char*)NULL, m_aSubPE);
    pDIB->SetPaletteEntries(0, 256, m_aSubPE);

    static int nPer=10;
    pDIB->ShiftRGBPercent(0, 256, nPer);
    nPer += 10;
    */

    // Create a new buffer and an identity palette
    if (!CreateOSB(pDIB))
        return FALSE;

    // Map the colors of the background DIB to
    // the identity palette we just created for the background in
    COSBView::Create(CDIB* pDIB)
    pDIB->MapColorsToPalette((CPalette*)GetOSBPalette());

    // Resize the main frame window to fit the background image
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit(FALSE); // Try shrinking first
    ResizeParentToFit(TRUE);  // Let's be daring

```

```

        // Render the entire scene to the off-screen buffer
        Render();
        // Paint the off-screen buffer to the window
        DrawOSB();

        return TRUE;
    }

    // A new sprite has been added to the document
    void CBMCView::NewSprite(CSprite* pSprite)
    {
        // CDIBPal pal;
        // pal.Load((char*)NULL, m_aSubPE);
        // pSprite->SetPaletteEntries(0, 256, m_aSubPE);
        static int n=1;
        pSprite->GetDIB()->RotatePaletteIndex(20, 4*4, n*4);
        pSprite->GetDIB()->RotatePaletteIndex(40, 4*4, n*4);
        n++;
        if (n > 4)
            n = 1;

        /*
        m_aSubPE[0].peRed = 0;
        m_aSubPE[0].peGreen = 0;
        m_aSubPE[0].peBlue = 0;
        m_aSubPE[0].peFlags = 0;
        m_pCapturedSprite->SetPaletteEntries(18, 1, m_aSubPE);
        */

        // map the colors in the sprite DIB to the
        // palette in the off-screen buffered view
        if (m_posBPal)
            pSprite->MapColorsToPalette((CPalette*)m_posBPal);

        // Render the scene
        Render();
        // Draw the new scene to the screen
        DrawOSB();
    }

    // Render the scene to the off-screen buffer
    // pClipRect defaults to NULL
    void CBMCView::Render(CRect* pClipRect)
    {
        CBMCDoc* pDoc = GetDocument();
        CRect rcDraw;

        // get the background DIB and render it
        CDIB* pDIB = pDoc->GetBackground();
        if (pDIB)
        {
            pDIB->GetRect(rcDraw);
            // If a clip rect was supplied use it
            if (pClipRect)
            {
                rcDraw.IntersectRect(pClipRect, &rcDraw);
            }
        }
    }

```



```

        // draw the image of the DIB to the os buffer
        ASSERT(m_posB);
        pDIB->CopyBits(m_posB,
                        rcDraw.left,
                        rcDraw.top,
                        rcDraw.right - rcDraw.left,
                        rcDraw.bottom - rcDraw.top,
                        rcDraw.left,
                        rcDraw.top);
    }

    // Render the sprite list from the bottom of the list to the top
    // Note that we always clip to the background DIB
    CSpriteList* pList = pDoc->GetSpriteList();
    POSITION pos = pList->GetTailPosition();
    CSprite* pSprite;
    while (pos)
    {
        pSprite = pList->GetPrev(pos);
        pSprite->Render(m_posB, &rcDraw);
    }
}

void CBMCView::OnLButtonDown(UINT nFlags, CPoint point)
{
    if (m_bMouseCaptured)
        return;

    CBMCDoc* pDoc = GetDocument();
    // See if it hit a sprite
    CSpriteList* pList = pDoc->GetSpriteList();
    CPhasedSprite* pSprite = (CPhasedSprite*)pList->HitTest(point);
    if (pSprite)
    {
        m_bMouseCaptured = TRUE;
        if (pSprite != m_pCapturedSprite)
        {
            m_pCapturedSprite = pSprite;
            TRACE("New Sprite %8.8XH\n", m_pCapturedSprite);
            CString* pstrFileBkg = pDoc->GetBackground()->GetName();
            CString* pstrFileSprite = pSprite->GetDIB()->GetName();
            CString strTitle;
            strTitle.Format("(B)%s, (S)%s", *pstrFileBkg,
                *pstrFileSprite);
            pDoc->SetTitle(strTitle);
        }
        SetCapture();
        m_ptOffset.x = point.x - m_pCapturedSprite->GetX();
        m_ptOffset.y = point.y - m_pCapturedSprite->GetY();
        ShowInfo(point, pSprite);
    }
}

// COSBView::OnLButtonDown(nFlags, point);
}

void CBMCView::OnLButtonUp(UINT nFlags, CPoint point)
{
    if (m_bMouseCaptured)

```

```

    {
        ::ReleaseCapture();
        m_bMouseCaptured = FALSE;
    }
// COSBView::OnLButtonUp(nFlags, point);
}

void CBMCView::OnMouseMove(UINT nFlags, CPoint point)
{
    if (m_bMouseCaptured)
    {
        ASSERT(m_pCapturedSprite);
        m_pCapturedSprite->SetLT(point.x - m_ptOffset.x,
                                   point.y - m_ptOffset.y);

        // Render and draw the changes
        RenderAndDrawDirtyList();
        CBMCDoc* pDoc = GetDocument();
        pDoc->SetModifiedFlag(TRUE);
    }
    ShowInfo(point, m_pCapturedSprite);
// COSBView::OnMouseMove(nFlags, point);
}

void CBMCView::ShowInfo(CPoint& point, CPhasedSprite* pSprite)
{
    // Show current state in the status bar
    CMainFrame* pFrame = (CMainFrame*)(AfxGetApp()->m_pMainWnd);
    ASSERT(pFrame);
    char buf[256];
    wsprintf(buf, "Cursor: %d,%d. ", point.x, point.y);
    if (pSprite)
    {
        wsprintf(&buf[strlen(buf)], "Sprite: %d,%d,%d Cell: %d,%d.",
                pSprite->GetX(),
                pSprite->GetY(),
                pSprite->GetZ(),
                pSprite->GetCellRow(),
                pSprite->GetCellColumn());
    }
    pFrame->SetStatusBarText(buf);
}

void CBMCView::ToggleOutline()
{
    if (!m_pCapturedSprite)
    {
        AfxMessageBox("Select the Sprite to toggle the outline.");
        return;
    }

// CDIBPal pal;
// pal.Load((char*)NULL, m_apPE[0]);

/*
    RGBQUAD* pQ = m_pCapturedSprite->GetClrTabAddress();
    if ((pQ[18].rgbRed == pQ[19].rgbRed) &&

```

```

        (pQ[18].rgbGreen == pQ[19].rgbGreen) &&
        (pQ[18].rgbBlue == pQ[19].rgbBlue))
    {
        pQ[18].rgbRed      = 255;
        pQ[18].rgbGreen    = 255;
        pQ[18].rgbBlue     = 0;
        pQ[18].rgbReserved = 0;
    }
    else
    {
        m_pCapturedSprite->ReplacePaletteIndexValue(18, 19);
    }
}

*/
/*
CClientDC dc(this);
TRIVERTEX      vert [4] ;
GRADIENT_TRIANGLE gTri;
vert [0] .x      = 0;
vert [0] .y      = 0;
vert [0] .Red    = 0x0000;
vert [0] .Green  = 0x0000;
vert [0] .Blue   = 0x0000;
vert [0] .Alpha  = 0x0000;
vert [1] .x      = 100;
vert [1] .y      = 0;
vert [1] .Red    = 0x0000;
vert [1] .Green  = 0x0000;
vert [1] .Blue   = 0xff00;
vert [1] .Alpha  = 0x0000;
vert [2] .x      = 100;
vert [2] .y      = 32;
vert [2] .Red    = 0x0000;
vert [2] .Green  = 0x0000;
vert [2] .Blue   = 0xff00;
vert [2] .Alpha  = 0x0000;
vert [3] .x      = 0;
vert [3] .y      = 32;
vert [3] .Red    = 0xff00;
vert [3] .Green  = 0xff00;
vert [3] .Blue   = 0xff00;
vert [3] .Alpha  = 0x0000;
gTRI[0].Vertex1  = 0
gTRI[0].Vertex2  = 1gTRI[0].Vertex3 = 2gTRI[1].Vertex1 = 0
gTRI[1].Vertex2  = 2gTRI[1].Vertex3 = 3
::GradientFill(dc.GetSafeHdc(), vert,4,&gTRI,1,GRADIENT_FILL_TRIANGLE);
*/

m_aSubPE[0].peRed += 10;
if (m_aSubPE[0].peRed > 255)
    m_aSubPE[0].peRed = 0;
m_aSubPE[0].peGreen += 10;
if (m_aSubPE[0].peGreen > 255)
    m_aSubPE[0].peGreen = 0;
m_aSubPE[0].peBlue += 10;
if (m_aSubPE[0].peBlue > 255)
    m_aSubPE[0].peBlue = 0;

m_pCapturedSprite->GetDIB()->SetPaletteEntries(18, 1, m_aSubPE);

```

BMCView.cpp

```
        NewSprite(m_pCapturedSprite);
    }

void CBMCView::OnVIEW640x480()
{
    CSize sizeTotal(640, 480);
    CDIB* pDIB;
    pDIB = new CDIB;
    pDIB->Create(sizeTotal.cx, sizeTotal.cy);
    SetScrollSizes(MM_TEXT, sizeTotal);
}

void CBMCView::OnVIEW800x600()
{
    // TODO: Add your command handler code here
}
```

BMCView.h

```
// BMCView.h : interface of the CBMCView class
//
///////////////////////////////////////////////////////////////////

#ifndef AFX_BMCVIEW_H_C193E4B0_88DD_11D1_ACEA_080009B9F339__INCLUDED_
#define AFX_BMCVIEW_H_C193E4B0_88DD_11D1_ACEA_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "../UC2Ani/OSBView.h"

class CSprite;
class CPhasedSprite;
class CBMCDDoc;

class CBMCView : public COSBView
{
protected: // create from serialization only
    CBMCView();
    DECLARE_DYNCREATE(CBMCView)

// Attributes
public:
    CBMCDDoc* GetDocument();
    CPhasedSprite* GetCapturedSprite() { return
m_pCapturedSprite; }

// Operations
public:
    BOOL NewBackground(CDIB* pDIB);
    virtual void Render(CRect* pClipRect=NULL);
    void NewSprite(CSprite* pSprite);

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CBMCView)
    public:
        virtual void OnDraw(CDC* pDC); // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void OnInitialUpdate();
    protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    }AFX_VIRTUAL

// Implementation
public:
    void ToggleOutline();
    virtual ~CBMCView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
#endif
```

BMCView.h

protected:

private:

```

        BOOL                m_bMouseCaptured; // TRUE if mouse captured
        CPhasedSprite*      m_pCapturedSprite; // Pointer to captured sprite
        (for drag)
        CPoint               m_ptOffset;        // offset into hit
sprite
        CPhasedSprite*      m_pMenuSprite;      // ptr to current sprite

//      typedef PALETTEENTRY* LPPALETTEENTRY;
//      LPPALETTEENTRY*      m_apPE;            // array of pointers to the
palette entries
//      int                  m_nPEs;
        PALETTEENTRY*       m_aSubPE;          // array of palette entries
(Substitutional palette)

        void SetSpriteZ(int z);
        void SetSpriteCols(int i);
        void SetSpriteRows(int i);
        void ShowInfo(CPoint& point, CPhasedSprite* pSprite);

```

// Generated message map functions

protected:

```

        //{AFX_MSG(CBMCView)
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        afx_msg void OnVIEW640x480();
        afx_msg void OnVIEW800x600();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

```

#ifndef _DEBUG // debug version in BMCView.cpp

inline CBMCDoc* CBMCView::GetDocument()

{ return (CBMCDoc*)m_pDocument; }

#endif

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}

// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif //

!defined(AFX_BMCVIEW_H__C193E4B0_88DD_11D1_ACEA_080009B9F339__INCLUDED_)

CloseDlg.cpp

```
// CloseDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "CloseDlg.h"
#include "ResMan.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

#ifdef _MALL
LPCTSTR BMP_CLOSE_BK          = "MCclose.bmp";
//const CRect      RECT_MESSAGE(47, 175, 367, 270);
//const CRect      RECT_TIME(98, 300, 321, 320);
const CRect      RECT_MESSAGE(47, 63, 280, 160);
const CRect      RECT_TIME(114, 145, 222, 165);
#else
LPCTSTR BMP_CLOSE_BK          = "U2Login|LoginBk.bmp";
LPCTSTR BMP_CLOSE_BTN_OK      = "U2Login|BtnOK.bmp";
LPCTSTR BMP_CLOSE_BTN_NO      = "U2Login|BtnNo.bmp";
const CRect      RECT_MESSAGE(47, 145, 367, 240);
const CRect      RECT_TIME(98, 270, 321, 290);
#endif

const int NUM_EPILOGUES=2;

//////////////////////////////////////
// CClosedlg dialog

CClosedlg::CClosedlg(CWnd* pParent /*=NULL*/)
: CDialog(CClosedlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CClosedlg)
    // m_strTimeReport = _T("");
    //}}AFX_DATA_INIT
    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    ASSERT(pMF);
    CTimeSpan tsElapsed = CTime::GetCurrentTime() - pMF->m_tmStart;
    CString strTime(tsElapsed.Format("%H:%M:%S"));
    m_strTimeReport.LoadString(IDS_TIME_REPORT);
    m_strTimeReport += strTime;

    m_nMsgCur = 0;

    m_NullBrush.CreateStockObject(NULL_BRUSH);
    m_ftMessage.CreateFont(-13, 0, 0, 0, FW_BOLD,    // NORMAL,
        FALSE, FALSE, 0,    // bItalic, bUnderline, cStrikeOut
```

```

        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH || FF_DONTCARE,
#ifdef _KOREAN
        "±¼.²Ã¼");
#else
        "Times New Roman");
#endif

    m_pPal = NULL;    // Set it NULL before loading DIB

    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + BMP_CLOSE_BK);
    m_pDIBBack = new CDIB;
    if (!m_pDIBBack->Load(strFile))
    {
        delete m_pDIBBack;
        m_pDIBBack = NULL;
        return;
    }

    if (pMF->Is256Palette())
    {
        // Use mainframe's palette to avoid color flickering
        m_pPal = pMF->GetPalette();
        m_pDIBBack->MapColorsToPalette(m_pPal);
        m_bPaletteCreated = FALSE;
    }
    else // Use original palette in the file for TRUE color system
    {
        // Create the palette from the DIB.
        CDIBPal* pDIBPal;
        pDIBPal = new CDIBPal;
        ASSERT(pDIBPal);
        if (!pDIBPal->Create(m_pDIBBack))
        {
            AfxMessageBox("Failed to create palette from DIB file");
            delete pDIBPal;
        }
        m_pPal = pDIBPal; // type casting to parent class
        m_bPaletteCreated = TRUE;
    }

#ifdef _MALL
    strFile = strPath + BMP_CLOSE_BTN_OK;
    m_btnOK.Load(strFile);
    m_btnOK.SetPalette(m_pPal);

    strFile = strPath + BMP_CLOSE_BTN_NO;
    m_btnCancel.Load(strFile);
    m_btnCancel.SetPalette(m_pPal);
#endif
}

CClosedDlg::~CClosedDlg()
{
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal && m_bPaletteCreated)

```


CloseDlg.cpp

```

        delete m_pPal;
    }

void CCloseDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCloseDlg)
    DDX_Text(pDX, IDC_ST_MESSAGE, m_strMessage);
    DDX_Text(pDX, IDC_ST_TIME_REPORT, m_strTimeReport);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCloseDlg, CDialog)
    //{{AFX_MSG_MAP(CCloseDlg)
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
    ON_WM_SIZE()
    ON_WM_ERASEBKGD()
    ON_WM_CTLCOLOR()
    ON_WM_NCHITTEST()
    ON_WM_RBUTTONDOWN()
    ON_WM_LBUTTONDOWNCLK()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CCloseDlg message handlers

void CCloseDlg::InitControl(const int nCtrlID, const CRect& rcCtrl)
{
    CWnd* pW = GetDlgItem(nCtrlID);
    ASSERT(pW);
    pW->SetWindowPos(NULL, rcCtrl.left, rcCtrl.top,
                     rcCtrl.Width(), rcCtrl.Height(),
                     SWP_NOZORDER | SWP_NOACTIVATE);
    pW->SetFont(&m_ftMessage, FALSE);    // do not Redraw
}

BOOL CCloseDlg::OnInitDialog()
{
    if (!m_pDIBBack)
        return FALSE;
    CDialog::OnInitDialog();

#ifdef _MALL
    m_btnOK.SubclassDlgItem(IDOK, this);
    m_btnCancel.SubclassDlgItem(IDCANCEL, this);

    CPoint ptLT(349, 238);
    m_btnOK.MoveResize(ptLT);
    ptLT.x = 17;
    m_btnCancel.MoveResize(ptLT);
#endif

    InitControl(IDC_ST_MESSAGE,      RECT_MESSAGE);
    InitControl(IDC_ST_TIME_REPORT,  RECT_TIME);
}

```

CloseDlg.cpp

```

        return TRUE; // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
    }

void CCloseDlg::OnPaletteChanged(CWnd* pFocusWnd)
{
    CDialog::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CCloseDlg::OnQueryNewPalette()
{
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE); //
foreground
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
        // if (u)
        // { // Some colors changed so we need to do a repaint.
        //     Invalidate(); // Repaint the lot.
        //     return TRUE; // Say we did something.
        // }
        return FALSE; // Say we did nothing.
    }
}

void CCloseDlg::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);

    if (m_pDIBBack)
    {
        SetWindowPos(NULL, 0, 0, m_pDIBBack->GetWidth(), m_pDIBBack-
>GetHeight(),
                                SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
    }
}

BOOL CCloseDlg::OnEraseBkgnd(CDC* pDC)
{
    CDialog::OnEraseBkgnd(pDC);

    // Make sure we have what we need to do a paint.
    if (!m_pDIBBack)
    {
        TRACE("CLoginDlg: No DIB!\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;

```

CloseDlg.cpp

```

        if (m_pPal)
        {
            pPalOld = pDC->SelectPalette(m_pPal, FALSE);    //
bForceBackground = FALSE
            // dc.RealizePalette();    // we realize in response to
WM_QUERYNEWPALETTE
        }
        m_pDIBBack->Draw(pDC, 0, 0);
        // Select old palette if we altered it.
        if (pPalOld)
            pDC->SelectPalette(pPalOld, FALSE);

        return TRUE;
    }

HBRUSH CCloseDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    pDC->SetBkMode(TRANSPARENT);
    return (HBRUSH)m_NullBrush;
}

UINT CCloseDlg::OnNcHitTest(CPoint point)
{
    UINT nHitTest = CDialog::OnNcHitTest(point);
#ifdef _MALL
    if ((nHitTest == HTCLIENT) && (::GetAsyncKeyState(MK_LBUTTON) < 0))
        nHitTest = HTCAPTION;
#endif
    return nHitTest;
}

void CCloseDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
    if (nFlags & MK_CONTROL)
    {
        // Authority Check
        if (RECT_MESSAGE.PtInRect(point))
        {
            UINT nID = IDS_EPILOGUE00 + m_nMsgCur;
            if (m_nMsgCur++ == NUM_EPILOGUES)
                m_nMsgCur = 0;
            m_strMessage.LoadString(nID);
            UpdateData(FALSE);
            InvalidateRect(RECT_MESSAGE, TRUE);
        }
    }
    CDialog::OnRButtonDown(nFlags, point);
}

void CCloseDlg::OnLButtonDblClk(UINT nFlags, CPoint point)
{
#ifdef _MALL
    CDialog::OnOK();
#else
    CDialog::OnLButtonDblClk(nFlags, point);
#endif
}

```

CloseDlg.h

```
#if !defined(AFX_CLOSEDLG_H__11F28E44_B405_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_CLOSEDLG_H__11F28E44_B405_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// CloseDlg.h : header file
//
#include "UC2Ani/PSButton.h"

////////////////////////////////////
// CClosedlg dialog
class CDIB;

class CClosedlg : public CDialog
{
// Construction
public:
    CClosedlg(CWnd* pParent = NULL);    // standard constructor
    virtual ~CClosedlg();

// Dialog Data
   //{{AFX_DATA(CClosedlg)
    enum { IDD = IDD_CLOSE };
    CString    m_strMessage;
    CString    m_strTimeReport;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CClosedlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    void InitControl(const int nCtrlID, const CRect& rcCtrl);

    CDIB*            m_pDIBBack;        // Background frame image
    CPalette*        m_pPal;            // main palette
    BOOL            m_bPaletteCreated;

#ifdef _MALL
    CPSButton        m_btnOK;
    CPSButton        m_btnCancel;
#endif

    CBrush            m_NullBrush;
    CFont            m_ftMessage;
    int              m_nMsgCur;        // Current Message index

// Generated message map functions
   //{{AFX_MSG(CClosedlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    afx_msg BOOL OnQueryNewPalette();
    afx_msg void OnSize(UINT nType, int cx, int cy);
    //}}AFX_MSG
};
```

CloseDlg.h

```
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
afx_msg UINT OnNcHitTest(CPoint point);
afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_CLOSEDLG_H__11F28E44_B405_11D1_80E2_080009B9F339__INCLUDED_)
```

```

//
// ConMan.cpp
//
// Connection Manager Source File
//
// Copyright Quarterview 1999
// joseph j. kim
//

#include <stdafx.h>
#include <afxmt.h>

#include "blocksock.h"
#include "httpUtility.h"
// #include "httpServerThread.h"

#include "ResMan.h"
#include "conman.h"
#include "UC2Doc.h"
#include "UC2Messages.h"
#include "Parser.h"
#include "MainFrm.h"
#include "PSOtherInfo.h"

#define SERVERMAXBUF      5000
#define MAXLINELENGTH    100

// #define USE_TRANSMITFILE // uncomment if you have Windows NT

extern CResMan gResMan;
extern CParser gParser;
extern CConMan gConMan;

// extern BOOL Parse(char* pStr, char** ppToken1, char** ppToken2);

// CEvent
// BOOL
//
// CBlockingSocket
// int volatile
// static CString
// CString
// //volatile CString
// CString
// CString
// //static int
// #define MAXBUF 50000

// CString g_strIPClient;
// CString g_strProxy = "ITGPROXY";
// BOOL g_bUseProxy = FALSE;

/*CString g_strIPServer;
BOOL g_bListening;
*/

```

```

char htmlHdrErr[] = "HTTP/1.0 404 Object Not Found\r\n"
                  "Server: Quarterview SOCK01\r\n"
                  "Content-Type: text/html\r\n"
                  "Accept-Ranges: bytes\r\n"
                  "Content-Length: 66\r\n\r\n" // WinInet wants
correct length

```

```

" <html><h1><body>QV/1.0 404 Object Not
Found</h1></body></html>\r\n";

```

```

char htmlHdrFmt[] = "HTTP/1.0 200 OK\r\n"
                  "Server: Quarterview SockServer\r\n"
                  "Date: %s\r\n"
                  "Content-Type: text/html\r\n"
                  "Accept-Ranges: bytes\r\n"
                  "Content-Length: %d\r\n";

```

```

char htmlHdrEnd[] = "</body></html>\r\n\r\n";

```

```

static BOOL htmlParseHdr(char* pStr, char** ppToken1, char** ppToken2)
// really stupid parsing routine
// (must find two tokens, each followed by a space)
{

```

```

    *ppToken1 = pStr;
    char* pch = strchr(pStr, ' ');
    if(pch) {
        *pch = '\0';
        pch++;
        *ppToken2 = pch;
        pch = strchr(pch, ' ');
        if(pch) {
            *pch = '\0';
            return TRUE;
        }
    }

```

```

    return FALSE;
}

```

```

static void htmlLogRequest(LPVOID pParam, char* pch, CSockAddr sa)
{
    // pParam holds the HWND for the destination window (in another thread)
    CString strGmt = CTime::GetCurrentTime().FormatGmt("%m/%d/%y %H:%M:%S
GMT");

```

```

    char text1[1000];
    wsprintf(text1, "SERVER CONNECTION # %d: IP addr = %s, port = %d --
%s\r\n",

```

```

        g_nConnection, sa.DottedDecimal(), sa.Port(), (const char*)
strGmt);

```

```

    strcat(text1, pch);
    ::SendMessage((HWND) pParam, EM_SETSEL, (WPARAM) 65534, 65535);
    ::SendMessage((HWND) pParam, EM_REPLACESEL, (WPARAM) 0, (LPARAM)
text1);
}

```

```

static CFile* htmlOpenFile(const char* pName)
{

```

```

    // if it's really a directory, open the default HTML file

```

```

CFileException e;
CFile* pFile = new CFile();
if(*pName == '/') pName++;
CString strName = pName;
if(pFile->Open(strName, CFile::modeRead, &e)) {
    return pFile;
}
if((e.m_cause == CFileException::accessDenied) ||
    (e.m_cause == CFileException::badPath)) { // directory?
    int nLength;
    // add a / unless it's the "root" directory
    if((nLength = strName.GetLength()) > 1) {
        if(strName[nLength - 1] != '/') {
            strName += '/';
        }
    }
    strName += g_strDefault;
    if(pFile->Open(strName, CFile::modeRead, &e)) {
        return pFile;
    }
}
delete pFile;
return NULL;
}

CConMan::CConMan()
{
    TRACE0("CConMan::CConMan()\n");

    g_bListening = FALSE;

    m_SockInfo.nPortServer = (USHORT) 800;
}

CConMan::~CConMan()
{
    TRACE0("CConMan::~CConMan()\n");
}

// this is actually not a real http server.
// it's a simplified version that runs on port 800.
// we use it for client to client and client to DAS communication.
// in the future i want this to be used to transfer html files
// to our browser and for mini-browsers off of the "home". -jjk
void CConMan::StartHttpServer()
{
    TRACE("\nCConMan::StartHttpServer()");

    try {
        CSocketAddr saServer;
        if (m_strIPServer.IsEmpty()) { // first or only IP

```



```

        saServer = CSocketAddr(INADDR_ANY,
                                (USHORT) m_SockInfo.nPortServer);
    }
    else { // if our computer has multiple IP addresses...
        saServer = CSocketAddr(m_strIPServer,
                                (USHORT) m_SockInfo.nPortServer);
    }

    g_sListen.Create();
    g_sListen.Bind(saServer);
    g_sListen.Listen();// start listening

    g_bListening = TRUE;
    g_nConnection = 0;

//
    g_strServerName = "das.quarterview.com";
    g_strServerIP = "203.229.183.82"; // syc 0514

    // this should be code to register with the DAS and change status
    // to "active" for other users using home presence window.

    m_SockInfo.nCmd = CHK_VERSION;

    try {

        AfxBeginThread(CConMan::DasCallThread,
                        (LPVOID) &m_SockInfo, THREAD_PRIORITY_NORMAL);
    }
    catch (CBlockingSocketException* e) {
        //AfxMessageBox(e);
        e->Delete();
    }

    CWinThread *lpThread =
        AfxBeginThread(CConMan::HttpAnswerThread,
                        (LPVOID) &m_SockInfo);
    m_hCallAnswerThread = lpThread->m_hThread;
}
catch(CBlockingSocketException* e) {
    g_sListen.Cleanup();

    TCHAR szCause[255];
    CString strFormatted;

    e->GetErrorMessage(szCause, 255);
    strFormatted = _T("QtttpServerError: ");
    strFormatted += szCause;

    AfxMessageBox(strFormatted);

    // LogBlockingSocketException(GetSafeHwnd(), "VIEW:", e);
    e->Delete();
}

```

```

}

void CConMan::StopHttpServer()
{
    TRACE("CConMan::StopHttpServer()");

    try {
        if (g_bListening) {
            g_sListen.Close();
        }
    }
    catch(CBlockingSocketException* e) {
//        LogBlockingSocketException(GetSafeHwnd(), "VIEW:", e);
        AfxMessageBox("\nStopHttpServer Error");
        e->Delete();
    }
}

UINT CConMan::DasCallThread(LPVOID pParam)
{
    TRACE("\nDasCallThread()");

    SOCKET_INFO* pSockInfo = (SOCKET_INFO *) pParam;
    // sends a blind request, followed by a request for a specific URL
    CHttpBlockingSocket sClient;

    char* buffer = new char[MAXBUF];
    int nBytesReceived = 0;

    char resp1[MAXLINELENGTH];
    //, resp2[MAXLINELENGTH];

    char qry_req[] = "QRY %s%s%s QTTP/1.0\r\n";
    char reg_req[] = "REG %s%s%s QTTP/1.0\r\n";
    char ver_req[] = "VER %s%s%s QTTP/1.0\r\n";

    char upd_req[] = "UPD %s%s%s QTTP/1.0\r\n";

    char headers[] =
        "User-Agent: Mozilla/1.22 (Windows; U; 32bit)\r\n"
        "Accept: */*\r\n"
//        "Accept: image/gif\r\n"
//        "Accept: image/x-xbitmap\r\n"
//        "Accept: image/jpeg\r\n"
// following line tests server's ability to not send the URL
//        "If-Modified-Since: Wed, 11 Sep 1996 20:23:04 GMT\r\n"
        "\r\n"; // need this

    CSockAddr saServer, saPeer, saTrace, saClient;
    try {
        sClient.Create();
        if (!g_strIPClient.IsEmpty()) {

```

```

        // won't work if network is assigning us our IP address
        // good only for intranets where client computer has
several IP addresses
        saClient = CSocketAddr(g_strIPClient);
        sClient.Bind(saClient);
    }
    if(g_bUseProxy) {
        saServer = CBlockingSocket::GetHostByName(g_strProxy,
(USHORT) pSockInfo->nPortServer);
    }
    else {
        if(g_sDasIP.IsEmpty()) {
            saServer =
CBlockingSocket::GetHostByName(g_strServerName, (USHORT) pSockInfo-
>nPortServer);
        }
        else {
            saServer = CSocketAddr(g_sDasIP, (USHORT) pSockInfo-
>nPortServer);
        }
    }
    sClient.Connect(saServer);
    sClient.GetSockAddr(saTrace);

    g_myIpAddr = saTrace.DottedDecimal();
    TRACE("SOCK CLIENT: GetSockAddr = %s, %d\n",
saTrace.DottedDecimal(), saTrace.Port());

/*
    if(g_bUseProxy) {
        wsprintf(buffer, request, "http://" , (const char*)
g_strServerName, g_strReq);
    }
    else {
        wsprintf(buffer, request, "", "", g_strReq);
    }
*/

    CString sResp;
    CString sMsg;

    switch (pSockInfo->nCmd) {

    case REG_ONLINE:

        TRACE("\nDasCallThread() - AUTH_ALIAS");

        sMsg.Format("%d`%d`%s`%s`", pSockInfo->nCmd, pSockInfo-
>nID, pSockInfo->sAlias, pSockInfo->sPass);
        wsprintf(buffer, reg_req, "", "", sMsg);

        {
            int nTry=4;

write_online:

            sClient.Write(buffer, strlen(buffer), 10);
            TRACE("\n wrote out %s", buffer);

```

ConMan.cpp

```

//      sClient.Write(headers, strlen(headers), 10);
//      TRACE("\n      wrote out %s", headers);

      nBytesReceived = sClient.ReadHttpHeaderLine(resp1,
MAXBUF, 8);

      TRACE("\nDAS RETURN1: %s", resp1);

      if (nBytesReceived==0 && nTry>0) {
          nTry--;
          goto write_online;
      }
      else if (nTry==0) {
          AfxMessageBox("Unable to register with DAS.
Other Users will not be able\n"
                        "to detect your online presence.");
      }

      TRACE("\nCConMan::DasCallThread -> No Response...
Bailing out.");

      sClient.Close();
      delete [] buffer;

      return NO_RESP;
    }

    gParser.CopyBuffer((CHAR*)resp1);

    gParser.GetValueRightToken(sResp, ARG_TOKEN);
    gParser.GetValueRightToken(sMsg, ARG_TOKEN);

    if (sResp == "RESP") {
        if (sMsg == "ONLINE_ACK") {
            TRACE("\nCConMan::DasCallThread -> OnlineACK");
            sClient.Close();
            delete [] buffer;

            return ONLINE_ACK;
        }
        else {
            TRACE("\nCConMan::DasCallThread -> NEG ACK");
            sClient.Close();
            delete [] buffer;

            return NEG_ACK;
        }
    }

    TRACE("\nCConMan::DasCallThread -> NO RESPONSE");
    return NO_RESP;

    sClient.Close();
    delete [] buffer;

```

```

        break;

    case UPD_MYINFO:

        TRACE("\nDasCallThread() - AUTH_ALIAS");

        wsprintf(buffer, upd_req, "", "", pSockInfo->str);

        sClient.Write(buffer, strlen(buffer), 10);
        TRACE("\n    wrote out %s", buffer);

        /*
        nBytesReceived = sClient.ReadHttpHeaderLine(respl, MAXBUF,
        10);

        TRACE("\nDAS RETURN1: %s", respl);

        if (sResp == "RESP") {

            if (sMsg == "UPD_OK") {
                TRACE("\nCConMan::DasCallThread -> UpdateMyInfo
                OK");

                sClient.Close();
                delete [] buffer;

                return UPD_OK;
            }
            else {
                TRACE("\nCConMan::DasCallThread -> UpdateMyInfo
                NO");

                sClient.Close();
                delete [] buffer;

                return UPD_NO;
            }
        }

        */

        sClient.Close();
        delete [] buffer;

        return UPD_OK;

        break;

    case AUTH_ALIAS:

        TRACE("\nDasCallThread() - AUTH_ALIAS");

        // 1. send alias and password with alias_auth cmd
        // 2. if receive auth_ok return ok_pass. if not then return
        bad_pass

        sMsg.Format("%d`s`s`s", pSockInfo->nCmd, pSockInfo->
        >sAlias, pSockInfo->sPass);
        wsprintf(buffer, qry_req, "", "", sMsg);
        sClient.Write(buffer, strlen(buffer), 10);
        TRACE("\n    wrote out %s", buffer);

```

```

sClient.Write(headers, strlen(headers), 10);
TRACE("\n   wrote out %s", headers);

nBytesReceived = sClient.ReadHttpHeaderLine(resp1, MAXBUF,
10);

TRACE("\nDAS RETURN1: %s", resp1);

gParser.CopyBuffer((CHAR*)resp1);

gParser.GetValueRightToken(sResp, ARG_TOKEN);
gParser.GetValueRightToken(sMsg, ARG_TOKEN);

//      char* pToken1; char* pToken2;
//      if (htmlParseHdr(resp1, &pToken1, &pToken2)) {
//
//          if (!strcmp(pToken1, "RESP")) {
//
//              if (sResp == "RESP") {
//
//                  do { // eat the remaining headers
//                      sClient.ReadHttpHeaderLine(resp2,
//MAXLINELENGTH, 10);
//                      TRACE("DAS RETURN2: %s", resp2);
//                  }
//                  while(strcmp(resp2, "\r\n"));
//
//                  if (!strcmp(pToken2, "ALIASOK")) {
//
//                      if (sMsg == "ALIAS_OK") {
//                          TRACE("\nCConMan::DasCallThread -> AliasOK");
//                          sClient.Close();
//                          delete [] buffer;
//
//                          return ALIAS_OK;
//                      }
//                      else if (!strcmp(pToken2, "ALIASENDENY")) {
//                      else if (sMsg == "ALIAS_DENY") {
//                          TRACE("\nCConMan::DasCallThread -> AliasDeny");
//                          sClient.Close();
//                          delete [] buffer;
//
//                          return ALIAS_DENY;
//                      }
//                      else if (sMsg == "ALIAS_NO") {
//                          TRACE("\nCConMan::DasCallThread -> AliasNEW");
//                          sClient.Close();
//                          delete [] buffer;
//
//                          return ALIAS_NEW;
//                      }
//                      else {
//                          TRACE("\nCConMan::DasCallThread ->
AliasUnknown");
//                          sClient.Close();

```

```

        delete [] buffer;

        return ALIAS_UNKNOWN; // what is this?
    }
//
}

TRACE("\n ERR- skipped htmlParseHdr...");
sClient.Close();
delete [] buffer;

return ALIAS_UNKNOWN;

break;

case NEW_ALIAS:

    TRACE("\nDasCallThread() - NEW_ALIAS");

    sMsg.Format("%d`%s`%s`%s`", pSockInfo->nCmd, pSockInfo-
>sAlias, pSockInfo->sPass, pSockInfo->sEmail);
    wsprintf(buffer, qry_req, "", "", sMsg);
    sClient.Write(buffer, strlen(buffer), 10);
    TRACE("\n   wrote out %s", buffer);
    sClient.Write(headers, strlen(headers), 10);
    TRACE("\n   wrote out %s", headers);

    nBytesReceived = sClient.ReadHttpHeaderLine(respl, MAXBUF,
10);

    TRACE("\nDAS RETURN1: %s", respl);

    gParser.CopyBuffer((CHAR*)respl);

    gParser.GetValueRightToken(sResp, ARG_TOKEN);
    gParser.GetValueRightToken(sMsg, ARG_TOKEN);

    if (sResp == "RESP") {

        if (sMsg == "ALIAS_APPROVE") {
            TRACE("\nCConMan::DasCallThread -> AliasOK");
            sClient.Close();
            delete [] buffer;

            return ALIAS_APPROVE;
        }
        else if (sMsg == "ALIAS_TAKEN") {
            TRACE("\nCConMan::DasCallThread -> AliasDeny");
            sClient.Close();
            delete [] buffer;

            return ALIAS_TAKEN;
        }
        else {
            TRACE("\nCConMan::DasCallThread ->
AliasUnknown");

            sClient.Close();
            delete [] buffer;

```

```

        return ALIAS_DENY;
    }
}

TRACE("\n ERR- skipped htmlParseHdr...");

sClient.Close();
delete [] buffer;
return ALIAS_UNKNOWN;

break;

case QRY_USERINFO:

    TRACE("\nDasCallThread() - QRY_USERINFO");

    // 1. send alias and password with user_personal cmd
    // 2. if receive user_personal info return ok_pass. if not
    then return bad_pass

    sMsg.Format("%d`s", pSockInfo->nCmd, pSockInfo->sAlias);
    wsprintf(buffer, qry_req, "", "", sMsg);
    sClient.Write(buffer, strlen(buffer), 10);
    TRACE("\n wrote out %s", buffer);
    sClient.Write(headers, strlen(headers), 10);
    TRACE("\n wrote out %s", headers);

    nBytesReceived = sClient.ReadHttpHeaderLine(respl, MAXBUF,
10);

    TRACE("\nDAS RETURN1: %s", respl);

    gParser.CopyBuffer((CHAR*)respl);

    gParser.GetValueRightToken(sResp, ARG_TOKEN);
    gParser.GetValueRightToken(sMsg, ARG_TOKEN);

    if (sResp == "RESP") {

        TRACE("\nCConMan::DasCallThread -> QRY_USERINFO OK");

        CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
        CUC2Doc* pDoc = (CUC2Doc*)pMF->GetActiveDocument();

        CPSOtherInfo      psoi(pDoc, PS_OTHERINFO_PAGE,
pSockInfo->sAlias);

        psoi.m_OtherInfoPage.m_sAlias = pSockInfo->sAlias;
        psoi.SetMemberInfo(&sMsg);
        psoi.DoModal();

        sClient.Close();
        delete [] buffer;

        return INFO_OK;
    }
}

```



```

    }

    TRACE("\n ERR- skipped htmlParseHdr...");
    sClient.Close();
    delete [] buffer;

    return INFO_NO;

    break;

case CHK_VERSION:

    TRACE("\nDasCallThread() - CHK_VERSION");

    sMsg.Format("%d`", pSockInfo->nCmd);
    wsprintf(buffer, ver_req, "", "", sMsg);
    sClient.Write(buffer, strlen(buffer), 10);
    TRACE("\n wrote out %s", buffer);

    // call some version checker function here...
    // or set NEED UPDATE variable to true...

    sClient.Close();
    delete [] buffer;
    return 0;

    break;

case FIND_ALIAS:
    break;

case FIND_BY_FIELD:
    break;

default:
    ;

}

sClient.Write(buffer, strlen(buffer), 10);
sClient.Write(headers, strlen(headers), 10);
// read all the server's response headers
do {
    nBytesReceived = sClient.ReadHttpHeaderLine(buffer, MAXBUF,
10);

    TRACE("SOCK CLIENT: %s", buffer);
} while(strcmp(buffer, "\r\n"));
// read the server's file
nBytesReceived = sClient.ReadHttpResponse(buffer, MAXBUF, 10);
TRACE("SOCK CLIENT: bytes received = %d\n", nBytesReceived);
if(nBytesReceived == 0) {
    AfxMessageBox("No response received. Bad URL?");
}

```

```

    }
    else {
        buffer[nBytesReceived] = '\0';
        ::MessageBox(::GetTopWindow(::GetDesktopWindow()), buffer,
"WINSOCK CLIENT", MB_OK);
    }

    // could do another request on sClient by calling Close, then
    Create, etc.
    }
    catch(CBlockingSocketException* e) {
        LogBlockingSocketException(pParam, "CLIENT:", e);
        e->Delete();
    }
    sClient.Close();
    delete [] buffer;
    return 0;
}

UINT CConMan::HttpCallThread(LPVOID pParam)
{
    TRACE("\nCConMan::HttpCallThread()");

    SOCKET_INFO* pSockInfo = (SOCKET_INFO *) pParam;

    // sends a blind request, followed by a request for a specific URL
    CHttpBlockingSocket sClient;
    char* buffer = new char[MAXBUF];
    int nBytesReceived = 0;
    // We're doing a blind GET, but we must provide server name if we're
    using a proxy.
    // A blind GET is supposed to retrieve the server's default HTML
    document.
    // Some servers don't have a default document but return a document
    name in the Location header.
    char request[] = "GET %s%s%s HTTP/1.0\r\n";
    char headers[] =
        "User-Agent: Mozilla/1.22 (Windows; U; 32bit)\r\n"
        "Accept: */*\r\n"
        "Accept: image/gif\r\n"
        "Accept: image/x-xbitmap\r\n"
        "Accept: image/jpeg\r\n"
    // following line tests server's ability to not send the URL
    //
    "If-Modified-Since: Wed, 11 Sep 1996 20:23:04 GMT\r\n"
    "\r\n"; // need this
    CSockAddr saServer, saPeer, saTest, saClient;
    try {

        sClient.Create();
        if(!g_strIPClient.IsEmpty()) {
            // won't work if network is assigning us our IP address
            // good only for intranets where client computer has
            several IP addresses
            saClient = CSockAddr(g_strIPClient);
            sClient.Bind(saClient);
        }
    }
}

```

```

        if(g_bUseProxy) {
            saServer = CBlockingSocket::GetHostByName(g_strProxy, 800);
        }
        else {
            if(g_strServerIP.IsEmpty()) {
                saServer =
CBlockingSocket::GetHostByName(g_strServerName, pSockInfo->nPortServer);
            }
            else {
                saServer = CSockAddr(g_strServerIP, pSockInfo-
>nPortServer);
            }
        }

        sClient.Connect(saServer);
        sClient.GetSockAddr(saTest);
        TRACE("SOCK CLIENT: GetSockAddr = %s, %d\n",
saTest.DottedDecimal(), saTest.Port());

        if(g_bUseProxy) {
            wsprintf(buffer, request, "http://" , (const char*)
g_strServerName, g_strReq);
        }
        else {
            wsprintf(buffer, request, "", "", g_strReq);
        }

        sClient.Write(buffer, strlen(buffer), 10);
        sClient.Write(headers, strlen(headers), 10);

        // read all the server's response headers
        do {
            nBytesReceived = sClient.ReadHttpHeaderLine(buffer, MAXBUF,
10);

            TRACE("SOCK CLIENT: %s", buffer);
        } while(strcmp(buffer, "\r\n"));
        // read the server's file
        nBytesReceived = sClient.ReadHttpResponse(buffer, MAXBUF, 10);
        TRACE("SOCK CLIENT: bytes received = %d\n", nBytesReceived);

        if(nBytesReceived == 0) {
            AfxMessageBox("No response received. Bad URL?");
        }
        else {
            //
            //
            buffer[nBytesReceived] = '\0';
            ::MessageBox(::GetTopWindow(::GetDesktopWindow()), buffer,
"WINSOCK CLIENT", MB_OK);

            CFile cf;
            TRACE("SOCK CLIENT: Attempting to write file: %s",
pSockInfo->strFileName);

            /*
                CString path(*gResMan.GetResPath());
                path += "\\neighbor\\";
                TRACE("\n file saving to path: %s", path);
                ::SetCurrentDirectory(path);
            */

```

```

        */
//      ::SetCurrentDirectory(pSockInfo->strFilePath);
//      TRACE("\n %s", *gResMan.GetResPath());

        if (cf.Open(pSockInfo->strFileName, CFile::modeCreate |
CFile::modeWrite )) {
            try {
                //      cf.SeekToBegin();
                cf.Write(buffer, strlen(buffer));
                cf.Write(buffer, nBytesReceived);
                TRACE("\n File written");
            }
            catch (CFileException *e) {
                AfxMessageBox("File Write Error");
                e->Delete();
            }
//      ::SetCurrentDirectory(*gResMan.GetResPath());
        }

    }

    // could do another request on sClient by calling Close, then
    Create, etc.
    }
    catch(CBlockingSocketException* e) {
//      LogBlockingSocketException(pParam, "CLIENT:", e);
        AfxMessageBox("HttpCallThread Exception");
        e->Delete();
    }

    g_Continue = TRUE;

    sClient.Close();
    delete [] buffer;
    return 0;
}

UINT CConMan::HttpAnswerThread(LPVOID pParam)
{
    TRACE0("\nCConMan::HttpAnswerThread\n");

    CSockAddr saClient;
    CHttpBlockingSocket sConnect;
    char* buffer = new char[SERVERMAXBUF];
    char message[100], headers[500], request1[MAXLINELENGTH],
request2[MAXLINELENGTH];

    char hdrErr[] = "HTTP/1.0 404 Object Not Found\r\n"
        "Server: QuarterView ERROR SOCK01\r\n"
        "Content-Type: text/html\r\n"
        "Accept-Ranges: bytes\r\n"
        "Content-Length: 66\r\n\r\n" // WinInet wants
correct length

```

```

        "<html><h1><body>HTTP/1.0 404 Object Not
Found</h1></body></html>\r\n";

    char hdrFmt[] = "HTTP/1.0 200 OK\r\n"
        "Server: Quarterview Http Socket\r\n"
        "Date: %s\r\n"
        "Content-Type: text/html\r\n"
        "Accept-Ranges: bytes\r\n"
        "Content-Length: %d\r\n";

    char html1[] = "Welcome to QuarterView's Online Chat Service Version:
1.0\r\n"
        "Message of the Day\r\n"
        "\r\n"
        "Please enjoy our service\r\n"
        "\r\n";

    char html2[] = "
        -joseph j. kim (CEO,
Quarterview.com)\r\n\r\n";

    CString strGmtNow = CTime::GetCurrentTime().FormatGmt("%a, %d %b %Y
%H:%M:%S GMT");
    int nBytesSent = 0;

    CFile* pFile = NULL;

    try {
        if(!g_sListen.Accept(sConnect, saClient)) {
            // view or application closed the listing socket
            g_bListening = FALSE;
            delete [] buffer;
            return 0;
        }
        g_nConnection++;

        CString strHttpSrvrPath = *gResMan.GetResPath();
        strHttpSrvrPath += "html\\";
        ::SetCurrentDirectory(strHttpSrvrPath);

        AfxBeginThread(CConMan::HttpAnswerThread, pParam,
THREAD_PRIORITY_NORMAL);
        // read request from client
        sConnect.ReadHttpRequestLine(request1, MAXLINELENGTH, 10);

        //
        LogRequest(pParam, request1, saClient);

        char* pToken1; char* pToken2;
        if(htmlParseHdr(request1, &pToken1, &pToken2)) {
            if (!strcmp(pToken1, "GET")) {
                do { // eat the remaining headers
                    sConnect.ReadHttpRequestLine(request2,
MAXLINELENGTH, 10);
                    TRACE("SERVER: %s", request2);
                }
                while(strcmp(request2, "\r\n"));
            }
        }
    }

```

```

        if (!strcmp(pToken2, "/custom")) { // special
request
            // send a "custom" HTML page
            wsprintf(message, "Hi! you are connection #%d
on IP %s, port %d Time: %s",
                                g_nConnection,
saClient.DottedDecimal(), saClient.Port(), strGmtNow);
            wsprintf(headers, hdrFmt, (const char*)
strGmtNow, strlen(html1)
                                + strlen(message) +
strlen(html2));

            // no If-Modified
            strcat(headers, "\r\n"); // blank line
            sConnect.Write(headers, strlen(headers), 10);
            sConnect.Write(html1, strlen(html1), 10);
            sConnect.Write(message, strlen(message), 10);
            sConnect.Write(html2, strlen(html2), 10);
        }
        else if (strchr(pToken2, '?')) { // CGI request
            // Netscape doesn't pass function name in a GET
            TRACE("SERVER: CGI request detected %s\n",
pToken2);

            // could load and run the ISAPI DLL here
        }
        else { // must be a file
            // assume this program has already set the
default WWW directory
            if ((pFile = htmlOpenFile(pToken2)) != NULL) {

                TRACE("\n    Getting File: '%s'",
pToken2);

                CFileStatus fileStatus;
                pFile->GetStatus(fileStatus);
                CString strGmtMod =
fileStatus.m_mtime.FormatGmt("%a, %d %b %Y %H:%M:%S GMT");
                char hdrModified[50];
                wsprintf(hdrModified, "Last-Modified:
%s\r\n\r\n", (const char*) strGmtMod);
                DWORD dwLength = pFile->GetLength();
                // Date: , Content-Length:
                wsprintf(headers, hdrFmt, (const char*)
strGmtNow, dwLength);

                strcat(headers, hdrModified);
                nBytesSent = sConnect.Write(headers,
strlen(headers), 10);

                TRACE("SERVER: header characters sent =
%d\n", nBytesSent);

                // would be a good idea to send the file
                // were less than the file's m_mtime
                nBytesSent = 0;

                if (::TransmitFile(sConnect, (HANDLE)
                                0, NULL, NULL,
TF_DISCONNECT)) {

```

```

        nBytesSent = (int) dwLength;
    }
#else
    DWORD dwBytesRead = 0;
    UINT uBytesToRead;
    // send file in small chunks (5K) to
    avoid big memory alloc overhead
    while(dwBytesRead < dwLength) {
        uBytesToRead = min(SERVERMAXBUF,
        dwLength - dwBytesRead);
        VERIFY(pFile->Read(buffer,
        uBytesToRead) == uBytesToRead);
        nBytesSent +=
        sConnect.Write(buffer, uBytesToRead, 10);
        dwBytesRead += uBytesToRead;
    }
#endif
    TRACE("SERVER: full file sent
    successfully\n");
    }
    else {
        nBytesSent = sConnect.Write(hdrErr,
        strlen(hdrErr), 10); // 404 Object Not Found
    }
}
else if(!strcmp(pToken1, "POST")) {
    do { // eat the remaining headers thru blank line
        sConnect.ReadHttpHeaderLine(request2,
        MAXLINELENGTH, 10);
        TRACE("SERVER: POST %s", request2);
    }
    while(strcmp(request2, "\r\n"));
    // read the data line sent by the client
    sConnect.ReadHttpHeaderLine(request2, MAXLINELENGTH,
    10);
    TRACE("SERVER: POST PARAMETERS = %s\n", request2);
    //
    LogRequest(pParam, request2, saClient);
    // launch ISAPI DLL here?
    nBytesSent = sConnect.Write(hdrErr, strlen(hdrErr),
    10); // 404 error for now
}
else {
    TRACE("SERVER: %s (not a GET or POST)\n", pToken1);
    // don't know how to eat the headers
}
}
else {
    TRACE("SERVER: bad request\n");
}
sConnect.Close(); // destructor can't close it
}
catch(CBlockingSocketException* pe) {
//
    LogBlockingSocketException(pParam, "SERVER:", pe);
    AfxMessageBox("httpanswerthread exception");
}

```

```
        pe->Delete();
    }
    TRACE("SERVER: file characters sent = %d\n", nBytesSent);
    delete [] buffer;
    if(pFile) delete pFile;

//    ::SetCurrentDirectory(*gResMan.GetResPath());

    return 0;

}
```


ConMan.h

```
//
// ConMan.h
//
// Connection Manager
//
// Copyright Quartermview 1999
//   joseph j. kim
//

#ifndef __CONMAN_H
#define __CONMAN_H

//extern CEvent g_eventContinue;
extern g_Continue;

typedef struct {

    int                nCmd;
    volatile USHORT    nPortServer;
    DWORD              dwLastError;
    volatile BOOL       bListening;

    CString            strFileName;    //file name to save file as
    CString            strFilePath;    // where to save the file.

    int                nID;
    CString            sAlias;
    CString            sPass;
    CString            sEmail;

    //    CString            sVersion;
    //    CString            str;

    /*    BOOL bCalling;
        BOOL bConnected;
        BOOL bAbort;
        char szOutgoingMessage[1000];
        char szHost[100];*/

} SOCKET_INFO;

typedef struct {
    CString    ip_addr;
    CString    nick;
} INVITE_INFO;

class CConMan : public CHttpBlockingSocket
{
public:
    CConMan();
    ~CConMan();
};
```

```

// methods

void StartHttpServer();
void StopHttpServer();

static UINT HttpAnswerThread(LPVOID);
static UINT HttpCallThread(LPVOID);
static UINT DasCallThread(LPVOID);

/*
  BOOL ParseHdr(char* pStr, char** ppToken1, char** ppToken2);
  void LogRequest(LPVOID pParam, char* pch, CSockAddr sa);
  CFile* OpenFile(const char* pName);
*/

// attributes

SOCKET_INFO          m_SockInfo;
INVITE_INFO          m_InviteInfo;
HANDLE               m_hCallAnswerThread;

CString              m_strIPServer;
CString              m_strDefault;

// operations

};

#endif // __CONMAN_H

```

```

        DWORD dwDU = 1500;
//      CUC2View* pView = GetUC2View();
        CString strText;

        LoadStage("0001demo");
        m_pStage->CreateActor(0, CPoint(5,11), TRUE, AS_STAND | DA_FR,
TRUE);

        CActor* pActor[10];
        pActor[0] = m_pStage->GetThisActor();
        ASSERT(pActor[0]);

        pActor[1] = m_pStage->CreateActor(10, CPoint(318,254), FALSE,
AS_STAND | DA_FR);
        Wait(dwDU);          if (!IsDemo())      break;
//      pActor->Act(CMD_HELLO);
        strText.LoadString(IDS_DEMO_1);
        pActor[1]->Chat(strText);
        DisplayText(strText);
        Wait(dwDU*2);          if (!IsDemo())      break;

        pActor[0]->SetState(AS_STAND | DA_BR);
        SendMoveCommand(CMD_MOVEF);
        Wait(dwDU*2);          if (!IsDemo())      break;
        strText.LoadString(IDS_DEMO_2);
        SendText(strText);
        Wait(dwDU*4);          if (!IsDemo())      break;

        pActor[2] = m_pStage->CreateActor(15, CPoint(195,127), FALSE,
AS_STAND | DA_FL);
        Wait(dwDU);          if (!IsDemo())      break;
        strText.LoadString(IDS_DEMO_3);
        pActor[2]->Chat(strText);
        DisplayText(strText);
        Wait(dwDU*3);          if (!IsDemo())      break;
        pActor[2]->Act(CMD_SMILE);

        SendMoveCommand(CMD_MOVEF);

        pActor[3] = m_pStage->CreateActor(7, CPoint(411,109), FALSE,
AS_STAND | DA_FR);
        Wait(dwDU);          if (!IsDemo())      break;
        strText = "Hi, Everybody! This is cool!";
        pActor[3]->Chat(strText);
        DisplayText(strText);
        Wait(dwDU*2);          if (!IsDemo())      break;

        SendMoveCommand(CMD_MOVEF);
        Wait(dwDU);          if (!IsDemo())      break;

        pActor[0]->SetDA(DA_FL, FALSE);
        strText.LoadString(IDS_DEMO_4);
        SendText(strText);
        Wait(dwDU*2);          if (!IsDemo())      break;
        SendCommand(CMD_MORPH);
        Wait(dwDU);          if (!IsDemo())      break;
        pActor[0]->SetDA(DA_FR, FALSE);
        Wait(dwDU);          if (!IsDemo())      break;

```

```

        SendMoveCommand(CMD_MOVEF);

        strText.LoadString(IDS_DEMO_5);
        SendText(strText);
        SendMoveCommand(CMD_MOVEF);

        strText.LoadString(IDS_DEMO_6);
        pActor[1]->Chat(strText);
        DisplayText(strText);

        OnBtnHistory();
        Wait(dwDU*5);                if (!IsDemo())    break;
// pView->ScrollToPosition(CPoint(80, 0));
        Wait(dwDU*2);                if (!IsDemo())    break;
        OnBtnHistory();
        Wait(dwDU);                  if (!IsDemo())    break;

        SendMoveCommand(CMD_MOVEF);
        Wait(dwDU);                  if (!IsDemo())    break;

        ////////////////////////////////////////////
        LoadStage("0002demo");
        m_pStage->CreateActor(0, CPoint(5,11), TRUE, m_wSavedState,
TRUE);

        pActor[0] = m_pStage->GetThisActor();
        Wait(dwDU*4);                if (!IsDemo())    break;

        SendCommand(CMD_STAND);
        strText.LoadString(IDS_DEMO_7);
        SendText(strText);
        SendMoveCommand(CMD_MOVEF);
        Wait(dwDU);                  if (!IsDemo())    break;

        pActor[1] = m_pStage->CreateActor(17, CPoint(225,113), FALSE,
AS_STAND | DA_FL);
        Wait(dwDU);                  if (!IsDemo())    break;

        SendMoveCommand(CMD_MOVEF);

        strText.LoadString(IDS_DEMO_8);
        pActor[1]->Chat(strText);
        DisplayText(strText);
        Wait(dwDU*4);                if (!IsDemo())    break;

        pActor[0]->SetState(AS_STAND | DA_FL);
        SendMoveCommand(CMD_MOVEF);

        pActor[1]->Act(CMD_MORPH);
        Wait(dwDU*3);                if (!IsDemo())    break;
        SendMoveCommand(CMD_MOVEF);
        pActor[1]->Act(CMD_STAND);

        pActor[2] = m_pStage->CreateActor(9, CPoint(355,334), FALSE);
        strText.LoadString(IDS_DEMO_9);
        pActor[2]->Chat(strText);
        DisplayText(strText);
        SendMoveCommand(CMD_MOVEF);

```

```

Wait(dwDU*3);          if (!IsDemo())    break;

pActor[0]->SetState(AS_STAND | DA_FR);
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;

pActor[3] = m_pStage->CreateActor(11, CPoint(128,254), FALSE,
AS_STAND | DA_FL);
Wait(dwDU);          if (!IsDemo())    break;
strText.LoadString(IDS_DEMO_10);
pActor[3]->Chat(strText);
DisplayText(strText);
SendMoveCommand(CMD_MOVEF);
Wait(dwDU*3);          if (!IsDemo())    break;

SendMoveCommand(CMD_MOVEF);
pActor[3]->Act(CMD_CRY);
Wait(dwDU);          if (!IsDemo())    break;
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;
pActor[3]->Act(CMD_MORPH);
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;
strText.LoadString(IDS_DEMO_11);
pActor[3]->Chat(strText);
Wait(dwDU);          if (!IsDemo())    break;
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;

////////////////////////////////////
LoadStage("0003demo");
m_pStage->CreateActor(0, CPoint(5,11), TRUE, m_wSavedState,
TRUE);

pActor[0] = m_pStage->GetThisActor();
Wait(dwDU*4);          if (!IsDemo())    break;

SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;

pActor[1] = m_pStage->CreateActor(12, CPoint(253,289), FALSE);
Wait(dwDU);          if (!IsDemo())    break;
strText.LoadString(IDS_DEMO_12);
pActor[1]->Chat(strText);
DisplayText(strText);
SendMoveCommand(CMD_MOVEF);
Wait(dwDU*8);          if (!IsDemo())    break;

strText.LoadString(IDS_DEMO_13);
SendText(strText);
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);          if (!IsDemo())    break;

CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();

```

```

CString strURL;
strURL.LoadString(IDS_UNICHAT_HOMEURL);
pMF->ShellBrowseURL(strURL);
Wait(dwDU*8);          if (!IsDemo())    break;
pMF->SetActiveWindow();

SendMoveCommand(CMD_MOVEF);
Wait(dwDU);           if (!IsDemo())    break;

SendMoveCommand(CMD_MOVEF);
Wait(dwDU);           if (!IsDemo())    break;

////////////////////////////////////
LoadStage("0001ctrm");
m_pStage->CreateActor(0, CPoint(5,11), TRUE, m_wSavedState,
TRUE);

pActor[0] = m_pStage->GetThisActor();
Wait(dwDU*4);          if (!IsDemo())    break;

pActor[0]->SetState(AS_STAND | DA_BR);
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);           if (!IsDemo())    break;
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);           if (!IsDemo())    break;
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);           if (!IsDemo())    break;
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);           if (!IsDemo())    break;

strText.LoadString(IDS_DEMO_14);
SendText(strText);
SendMoveCommand(CMD_MOVEF);
Wait(dwDU);           if (!IsDemo())    break;

ClearHistory();

```

```

// DlgAni.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "DlgAni.h"
#include "ResMan.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

//////////////////////////////////////
// CDlgAni dialog

CDlgAni::CDlgAni(LPCTSTR strBack, LPCTSTR strAni, const int nPages, CWnd*
pParent /*=NULL*/)
: CDialog(CDlgAni::IDD, pParent)
{
    //{{AFX_DATA_INIT(CDlgAni)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    m_nNumPages = nPages;
    m_nCurPage = 0;

    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + strBack);
    m_pPal = NULL; // initialize in case of failure
    m_pDIBBack = NULL;
    m_pPSAni = NULL;
    if (strBack)
    {
        m_pDIBBack = new CDIB;
        if (!m_pDIBBack->Load(strFile))
        {
            delete m_pDIBBack;
            m_pDIBBack = NULL;
            return;
        }
    }

    if (strAni)
    {
        strFile = strPath + strAni;
        m_pPSAni = new CPhasedSprite;
        if (!m_pPSAni->Load(strFile))
        {
            delete m_pPSAni;
            m_pPSAni = NULL;
            return;
        }
    }
}

```

```

    }
    m_pPSAni->SetNumCells(nPages, 1);
    if (!m_pDIBBack)
    {
        m_pDIBBack = m_pPSAni->GetDIB();
    }
}

CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
ASSERT(pMF);
if (pMF->Is256Palette())
{
    // Use mainframe's palette to avoid color flickering
    m_pPal = pMF->GetPalette();
    m_pDIBBack->MapColorsToPalette(m_pPal);
    m_bPaletteCreated = FALSE;
}
else // Use original palette in the file for TRUE color system
{
    // Create the palette from the DIB.
    CDIBPal* pDIBPal;
    pDIBPal = new CDIBPal;
    ASSERT(pDIBPal);
    if (!pDIBPal->Create(m_pDIBBack))
    {
        AfxMessageBox("Failed to create palette from DIB file");
        delete pDIBPal;
    }
    m_pPal = pDIBPal; // type casting to parent class
    m_bPaletteCreated = TRUE;
}
m_nCurPage = 0;

m_uiTimer = 0;
m_dwLastTick = 0L;
m_dwStartTick = 0L;

m_bAutoDestroy = FALSE;
m_dwDurationTick = 9000L; // 9 sec
m_uiElapse = 1000;
m_ptLTAni = CPoint(0, 0);
m_ptLeftTop = CPoint(200, 200);
}

CDlgAni::~CDlgAni()
{
    if (m_pDIBBack)
    {
        if (m_pPSAni)
        {
            if (m_pDIBBack != m_pPSAni->GetDIB())
                delete m_pDIBBack;
        }
        else
        {
            delete m_pDIBBack;
        }
    }
    if (m_pPSAni)

```



```

        delete m_pPSAni;

        if (m_pPal && m_bPaletteCreated)
            delete m_pPal;
    }

void CDlgAni::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlgAni)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgAni, CDialog)
    //{{AFX_MSG_MAP(CDlgAni)
        ON_WM_ERASEBKGND()
        ON_WM_PAINT()
        ON_WM_NCHITTEST()
        ON_WM_PALETTECHANGED()
        ON_WM_QUERYNEWPALETTE()
        ON_WM_TIMER()
        ON_WM_SIZE()
        ON_WM_DESTROY()
        ON_WM_LBUTTONDOWNCLK()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CDlgAni message handlers

BOOL CDlgAni::OnInitDialog()
{
    CDialog::OnInitDialog();

    if (!m_pDIBBack)
        return FALSE;
    CDialog::OnInitDialog();

    m_uiTimer = SetTimer(4004, m_uiElapse, NULL);    // CWnd::
    m_dwStartTick = ::GetTickCount();

    ClientToScreen(&m_ptLeftTop); // CWnd::
    SetWindowPos(NULL, m_ptLeftTop.x, m_ptLeftTop.y, 0, 0,
        SWP_NOSIZE | SWP_NOZORDER | SWP_NOACTIVATE);

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

BOOL CDlgAni::OnEraseBkgnd(CDC* pDC)
{
    CDialog::OnEraseBkgnd(pDC);

    // Make sure we have what we need to do a paint.

```

```

    if (!m_pDIBBack)
    {
        TRACE("CDlgAni: No DIB!\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = pDC->SelectPalette(m_pPal, FALSE);    //
bForceBackground = FALSE
        // dc.RealizePalette();    // we realize in response to
WM_QUERYNEWPALETTE
    }
    m_pDIBBack->Draw(pDC, 0, 0);

    // Select old palette if we altered it.
    if (pPalOld)
        pDC->SelectPalette(pPalOld, FALSE);

    return TRUE;
}

void CDlgAni::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    // Make sure we have what we need to do a paint.
    if (!m_pDIBBack)
    {
        TRACE("CDlgAni: No DIB!\n");
        return;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = dc.SelectPalette(m_pPal, FALSE);    //
bForceBackground = FALSE
        // dc.RealizePalette();    // we realize in response to
WM_QUERYNEWPALETTE
    }
    if (m_pPSAni)
    {
        m_pPSAni->SetCell(m_nCurPage);
        m_pPSAni->Draw(&dc, m_ptLTAni);
    }
    // Select old palette if we altered it.
    if (pPalOld)
        dc.SelectPalette(pPalOld, FALSE);
}

UINT CDlgAni::OnNcHitTest(CPoint point)
{
    UINT nHitTest = CDialog::OnNcHitTest(point);

```

```

        if ((nHitTest == HTCLIENT) && (::GetAsyncKeyState(MK_LBUTTON) < 0))
            nHitTest = HTCAPTION;
        return nHitTest;
    }

void CDlgAni::OnPaletteChanged(CWnd* pFocusWnd)
{
    CDialog::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CDlgAni::OnQueryNewPalette()
{
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE); //
foreground
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
        // if (u)
        // { // Some colors changed so we need to do a repaint.
        //     Invalidate(); // Repaint the lot.
        //     return TRUE; // Say we did something.
        // }
        return FALSE; // Say we did nothing.
    }
}

void CDlgAni::OnTimer(UINT nIDEvent)
{
    if (m_bAutoDestroy && (::GetTickCount() - m_dwStartTick) >
m_dwDurationTick)
        EndDialog(TRUE);
    if (m_nNumPages > 1)
        ShowNextPage();
// CDialog::OnTimer(nIDEvent);
}

void CDlgAni::ShowNextPage()
{
    m_nCurPage++;
    if (m_nCurPage >= m_nNumPages)
        m_nCurPage = 0;
    CRect rcAni;
    m_pPSAni->GetRect(rcAni);
    InvalidateRect(&rcAni, FALSE);
}

void CDlgAni::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);
}

```

```

        if (m_pDIBBack)
        {
            SetWindowPos(NULL, 0, 0, m_pDIBBack->GetWidth(), m_pDIBBack-
>GetHeight(),
                                SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
        }
    }

void CDlgAni::OnDestroy()
{
    CDialog::OnDestroy();

    if (m_uiTimer)
    {
        KillTimer(m_uiTimer);
        m_uiTimer = 0;
    }
}

void CDlgAni::OnLButtonDblClk(UINT nFlags, CPoint point)
{
#ifdef _MALL
    EndDialog(TRUE);
    // CDialog::OnOK();
#else
    CDialog::OnLButtonDblClk(nFlags, point);
#endif
}

```

DlgAni.h

```

#if !defined(AFX_DLGANI_H__D89C3A46_988E_11D2_89CC_0080C7EADFBB__INCLUDED_)
#define AFX_DLGANI_H__D89C3A46_988E_11D2_89CC_0080C7EADFBB__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgAni.h : header file
//

/////////////////////////////////////////////////////////////////
// CDlgAni dialog
class CDIB;
class CPhasedSprite;

class CDlgAni : public CDialog
{
// Construction
public:
    CDlgAni(LPCTSTR strBack, LPCTSTR strAni, const int nPages=1, CWnd*
pParent=NULL); // standard constructor
    ~CDlgAni();

    void SetAutoDestroy(const BOOL bAutoDestroy) { m_bAutoDestroy =
bAutoDestroy; }
    void SetTimeAttr(const UINT uiElapse, const DWORD dwDT)
    {
        m_uiElapse = uiElapse;
        m_dwDurationTick = dwDT;
    }
    void SetLT(const int nx, const int ny)
    {
        m_ptLTAni = CPoint(nx, ny);
    }
    void SetRelPosition(const int nX, const int nY)
    {
        m_ptLeftTop = CPoint(nX, nY);
    }

// Dialog Data
    //{AFX_DATA(CDlgAni)
    enum { IDD = IDD_DLG_ANI };
    // NOTE: the ClassWizard will add data members here
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CDlgAni)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    void ShowNextPage();

    CDIB* m_pDIBBack; // Background frame image

```

DlgAni.h

```

CPhasedSprite*   m_pPSAni;           // Animated Background
CPalette*        m_pPal;             // main palette
BOOL             m_bPaletteCreated;

int              m_nCurPage;
int              m_nNumPages;

UINT             m_uiTimer;
UINT             m_uiElapse;         // ticking period
DWORD            m_dwLastTick;
DWORD            m_dwStartTick;
DWORD            m_dwDurationTick;
BOOL             m_bAutoDestroy;
CPoint           m_ptLTAni;
CPoint           m_ptLeftTop;

// Generated message map functions
//{{AFX_MSG(CDlgAni)
virtual BOOL OnInitDialog();
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
afx_msg void OnPaint();
afx_msg UINT OnNcHitTest(CPoint point);
afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
afx_msg BOOL OnQueryNewPalette();
afx_msg void OnTimer(UINT nIDEvent);
afx_msg void OnSize(UINT nType, int cx, int cy);
afx_msg void OnDestroy();
afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_DLGANI_H__D89C3A46_988E_11D2_89CC_0080C7EADFBB__INCLUDED_)

```

DlgPDA.cpp

```
// DlgPDA.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "DlgPDA.h"
#include "ResMan.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

LPCTSTR BMP_PDA_BK = "MPDABk.bmp";
LPCTSTR BMP_PDA_BTN_CLOSE = "MPBtClos.bmp";
LPCTSTR BMP_PDA_BTN_BUY = "MPBtBuy.bmp";
LPCTSTR BMP_PDA_BTN_MORE = "MPBtMore.bmp";
LPCTSTR BMP_PDA_BTN_AUTO = "MPBtAuto.bmp";
LPCTSTR BMP_PDA_BTN_LEFT = "MPBtLeft.bmp";
LPCTSTR BMP_PDA_BTN_RIGHT = "MPBtRght.bmp";

const CPoint PTLT_ANI_PDA[2] = {CPoint(14, 8), CPoint(2, 151)};

////////////////////////////////////
// CDlgPDA dialog

CDlgPDA::CDlgPDA(LPCTSTR strUpper, LPCTSTR strLower, const int nPages, CWnd*
pParent /*=NULL*/)
: CDialog(CDlgPDA::IDD, pParent)
{
    //{AFX_DATA_INIT(CDlgPDA)
    // NOTE: the ClassWizard will add member initialization here
    //{AFX_DATA_INIT
    m_astrAniFile[0] = strUpper;
    m_astrAniFile[1] = strLower;
    m_nPages = nPages;

    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + BMP_PDA_BK);
    m_pDIBBack = new CDIB;
    m_pPal = NULL; // initialize in case of failure
    if (!m_pDIBBack->Load(strFile))
    {
        delete m_pDIBBack;
        m_pDIBBack = NULL;
        return;
    }

    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    ASSERT(pMF);
    if (pMF->Is256Palette())

```

```

{
    // Use mainframe's palette to avoid color flickering
    m_pPal = pMF->GetPalette();
    m_pDIBBack->MapColorsToPalette(m_pPal);
    m_bPaletteCreated = FALSE;
}
else // Use original palette in the file for TRUE color system
{
    // Create the palette from the DIB.
    CDIBPal* pDIBPal;
    pDIBPal = new CDIBPal;
    ASSERT(pDIBPal);
    if (!pDIBPal->Create(m_pDIBBack))
    {
        AfxMessageBox("Failed to create palette from DIB file");
        delete pDIBPal;
    }
    m_pPal = pDIBPal; // type casting to parent class
    m_bPaletteCreated = TRUE;
}

strFile = strPath + BMP_PDA_BTN_CLOSE;
m_btnClose.Load(strFile, 2);
m_btnClose.SetPalette(m_pPal);

strFile = strPath + BMP_PDA_BTN_BUY;
m_btnBuy.Load(strFile, 2);
m_btnBuy.SetPalette(m_pPal);

strFile = strPath + BMP_PDA_BTN_MORE;
m_btnMore.Load(strFile, 2);
m_btnMore.SetPalette(m_pPal);

strFile = strPath + BMP_PDA_BTN_AUTO;
m_btnAuto.Load(strFile, 2);
m_btnAuto.SetPalette(m_pPal);

strFile = strPath + BMP_PDA_BTN_LEFT;
m_btnLeft.Load(strFile, 2);
m_btnLeft.SetPalette(m_pPal);

strFile = strPath + BMP_PDA_BTN_RIGHT;
m_btnRight.Load(strFile, 2);
m_btnRight.SetPalette(m_pPal);

for (int i=0; i < 2; i++) // Upper and Lower
{
    strFile = strPath + m_astrAniFile[i];
    m_apDIBAni[i] = new CPhasedSprite;
    if (!m_apDIBAni[i]->Load(strFile))
    {
        delete m_apDIBAni[i];
        m_apDIBAni[i] = NULL;
        return; // skip
    }
    else
    {
        m_apDIBAni[i]->SetNumCells(1, m_nPages); // Row, Col
        m_apDIBAni[i]->SetCell(0);
    }
}

```



```

        m_apDIBAni[i] ->GetRect(m_arcAni[i]);
        m_arcAni[i].OffsetRect(PTLT_ANI_PDA[i]);
    }
}
m_nCurPage = 0;

m_uiTimer = 0;
m_dwLastTick      = 0L;
m_dwStartTick     = 0L;
m_bAutoDestroy    = FALSE;
m_bAutoPages      = FALSE;
m_uiInterval      = 1500;
m_dwDurationTick  = 9000L;      // 9 sec
}

CDlgPDA::~CDlgPDA()
{
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal && m_bPaletteCreated)
        delete m_pPal;
    for (int i=0; i < 2; i++)
    {
        if (m_apDIBAni[i])
            delete m_apDIBAni[i];
    }
}

void CDlgPDA::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlgPDA)
    // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgPDA, CDialog)
    //{{AFX_MSG_MAP(CDlgPDA)
    ON_BN_CLICKED(IDC_PDA_BTN_AUTO, OnPdaBtnAuto)
    ON_BN_CLICKED(IDC_PDA_BTN_BUY, OnPdaBtnBuy)
    ON_BN_CLICKED(IDC_PDA_BTN_LEFT, OnPdaBtnLeft)
    ON_BN_CLICKED(IDC_PDA_BTN_MORE, OnPdaBtnMore)
    ON_BN_CLICKED(IDC_PDA_BTN_RIGHT, OnPdaBtnRight)
    ON_WM_SIZE()
    ON_WM_ERASEBKGD()
    ON_WM_NCHITTEST()
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
    ON_WM_MOVE()
    ON_WM_PAINT()
    ON_WM_TIMER()
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CDlgPDA message handlers

```

```

BOOL CDlgPDA::OnInitDialog()
{
    if (!m_pDIBBack)
        return FALSE;
    CDialog::OnInitDialog();

    m_btnClose.SubclassDlgItem(IDOK, this);
    m_btnBuy.SubclassDlgItem(IDC_PDA_BTN_BUY, this);
    m_btnMore.SubclassDlgItem(IDC_PDA_BTN_MORE, this);
    m_btnAuto.SubclassDlgItem(IDC_PDA_BTN_AUTO, this);
    m_btnLeft.SubclassDlgItem(IDC_PDA_BTN_LEFT, this);
    m_btnRight.SubclassDlgItem(IDC_PDA_BTN_RIGHT, this);

    CPoint ptLT(132, 245);
    m_btnClose.MoveResize(ptLT);
    ptLT.x = 66;
    m_btnBuy.MoveResize(ptLT);
    ptLT.x = 0;
    m_btnMore.MoveResize(ptLT);
    ptLT = CPoint(63, 120);
    m_btnAuto.MoveResize(ptLT);
    ptLT.x = 0;
    m_btnLeft.MoveResize(ptLT);
    ptLT.x = 135;
    m_btnRight.MoveResize(ptLT);

    m_uiTimer = SetTimer(4003, m_uiInterval, NULL);
    m_dwStartTick = ::GetTickCount();

    ptLT = CPoint(435, 104);
    ClientToScreen(&ptLT); // CWnd::
    SetWindowPos(NULL, ptLT.x, ptLT.y, 0, 0,
        SWP_NOSIZE | SWP_NOZORDER | SWP_NOACTIVATE);
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CDlgPDA::PlayCameraSound()
{
    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + "Camera.wav");
    ::sndPlaySound(strFile, SND_ASYNC);
}

void CDlgPDA::OnPdaBtnAuto()
{
    m_bAutoPages = !m_bAutoPages;
}

void CDlgPDA::OnPdaBtnBuy()
{
}

void CDlgPDA::OnPdaBtnLeft()
{
    PlayCameraSound();
}

```

```

        m_nCurPage--;
        if (m_nCurPage < 0)
            m_nCurPage = m_nPages - 1;
        for (int i=0; i < 2; i++)
        {
            InvalidateRect(m_arcAni[i], FALSE);
        }
    }

void CDlgPDA::OnPdaBtnMore()
{
}

void CDlgPDA::OnPdaBtnRight()
{
    PlayCameraSound();
    m_nCurPage++;
    if (m_nCurPage >= m_nPages)
        m_nCurPage = 0;
    for (int i=0; i < 2; i++)
    {
        InvalidateRect(m_arcAni[i], FALSE);
    }
}

void CDlgPDA::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);

    if (m_pDIBBack)
    {
        SetWindowPos(NULL, 0, 0, m_pDIBBack->GetWidth(), m_pDIBBack->GetHeight(),
            SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
    }
}

BOOL CDlgPDA::OnEraseBkgnd(CDC* pDC)
{
    CDialog::OnEraseBkgnd(pDC);

    // Make sure we have what we need to do a paint.
    if (!m_pDIBBack)
    {
        TRACE("CLoginDlg: No DIB!\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = pDC->SelectPalette(m_pPal, FALSE);    //
bForceBackground = FALSE
        // dc.RealizePalette();    // we realize in response to
WM_QUERYNEWPALETTE
    }
}

```

```

        m_pDIBBack->Draw(pDC, 0, 0);
        // Select old palette if we altered it.
        if (pPalOld)
            pDC->SelectPalette(pPalOld, FALSE);

        return TRUE;
    }

void CDlgPDA::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    // Make sure we have what we need to do a paint.
    if (!m_pDIBBack) // || !m_pDIBOval
    {
        TRACE("CLoginDlg: No DIB!\n");
        return;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = dc.SelectPalette(m_pPal, FALSE); //
        bForceBackground = FALSE
        // dc.RealizePalette(); // we realize in response to
        WM_QUERYNEWPALETTE
    }
    for (int i=0; i < 2; i++)
    {
        if (m_apDIBAni[i])
        {
            m_apDIBAni[i]->SetCell(m_nCurPage);
            m_apDIBAni[i]->Draw(&dc, PTLT_ANI_PDA[i]);
        }
    }
    // Select old palette if we altered it.
    if (pPalOld)
        dc.SelectPalette(pPalOld, FALSE);
}

UINT CDlgPDA::OnNcHitTest(CPoint point)
{
    UINT nHitTest = CDialog::OnNcHitTest(point);
    if ((nHitTest == HTCLIENT) && (::GetAsyncKeyState(MK_LBUTTON) < 0))
        nHitTest = HTCAPTION;
    return nHitTest;
}

void CDlgPDA::OnPaletteChanged(CWnd* pFocusWnd)
{
    CDialog::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();
}

```

```

BOOL CDlgPDA::OnQueryNewPalette()
{
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE);    //
foreground
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
        // if (u)
        // {    // Some colors changed so we need to do a repaint.
        //     Invalidate(); // Repaint the lot.
        //     return TRUE; // Say we did something.
        // }
        return FALSE; // Say we did nothing.
    }
}

void CDlgPDA::OnMove(int x, int y)
{
    CDialog::OnMove(x, y);
}

void CDlgPDA::OnTimer(UINT nIDEvent)
{
    if (m_bAutoDestroy && (::GetTickCount() - m_dwStartTick) >
m_dwDurationTick)
        EndDialog(TRUE);
    if (m_bAutoPages)
        OnPdaBtnRight();
    // CDialog::OnTimer(nIDEvent);
}

void CDlgPDA::OnDestroy()
{
    CDialog::OnDestroy();

    if (m_uiTimer)
    {
        KillTimer(m_uiTimer);
        m_uiTimer = 0;
    }
}

```

DlgPDA.h

```

#if !defined(AFX_DLGPDA_H__A78B1261_47F3_11D2_BCFD_0080C7EADFBB__INCLUDED_)
#define AFX_DLGPDA_H__A78B1261_47F3_11D2_BCFD_0080C7EADFBB__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgPDA.h : header file
//
#include "UC2Ani/PSButton.h"

////////////////////////////////////
// CDlgPDA dialog
class CDIB;
class CPhasedSprite;

class CDlgPDA : public CDialog
{
// Construction
public:
    CDlgPDA(LPCTSTR strUpper, LPCTSTR strLower, const int nPages, CWnd*
pParent=NULL); // standard constructor
    ~CDlgPDA();

    void SetMode(const BOOL bAutoDestroy, const BOOL bAutoPages)
    {
        m_bAutoDestroy = bAutoDestroy;
        m_bAutoPages = bAutoPages;
    }
    void SetPage(const int nPage) { m_nCurPage = nPage; }
    void SetDuration(const UINT uiInt, const DWORD dwD)
    { m_uiInterval = uiInt; m_dwDurationTick = dwD; }

// Dialog Data
//{{AFX_DATA(CDlgPDA)
enum { IDD = IDD_DLG_PDA };
// NOTE: the ClassWizard will add data members here
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CDlgPDA)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
    void PlayCameraSound();

    CDIB* m_pDIBBack; // Background frame image
    CPhasedSprite* m_apDIBAni[2]; // Ani Upper, Lower
    CPalette* m_pPal; // main palette
    BOOL m_bPaletteCreated;

    CPSButton m_btnClose;
    CPSButton m_btnBuy;

```

DlgPDA.h

```

CPSButton      m_btnMore;
CPSButton      m_btnAuto;
CPSButton      m_btnLeft;
CPSButton      m_btnRight;
CRect          m_arcAni[2];

int            m_nCurPage;
UINT          m_uiTimer;
DWORD         m_dwLastTick;
DWORD         m_dwStartTick;
DWORD         m_dwDurationTick;
UINT          m_uiInterval;

BOOL          m_bAutoDestroy;
BOOL          m_bAutoPages;

CString        m_astrAniFile[2];
// CPoint      m_aprLT[2];
int           m_nPages;

// Generated message map functions
//{{AFX_MSG(CDlgPDA)
virtual BOOL OnInitDialog();
afx_msg void OnPdaBtnAuto();
afx_msg void OnPdaBtnBuy();
afx_msg void OnPdaBtnLeft();
afx_msg void OnPdaBtnMore();
afx_msg void OnPdaBtnRight();
afx_msg void OnSize(UINT nType, int cx, int cy);
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
afx_msg UINT OnNcHitTest(CPoint point);
afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
afx_msg BOOL OnQueryNewPalette();
afx_msg void OnMove(int x, int y);
afx_msg void OnPaint();
afx_msg void OnTimer(UINT nIDEvent);
afx_msg void OnDestroy();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_DLGPDA_H__A78B1261_47F3_11D2_BCFD_0080C7EADFBB__INCLUDED_)

```

```
#FILELIST=
{
u2res000.rit=0,4614,1998,4,1,10,22; // 0: do not extract
0000csin.sit=0,1811,1998,3,31,14,41;
0001ctrm.sit=0,2649,1998,4,1,11,10;
0001demo.sit=0,2454,1998,3,26,14,59;
0002ctrm.sit=0,1955,1998,4,1,11,10;
0002demo.sit=0,1851,1998,3,26,14,59;
0003ctrm.sit=0,2256,1998,4,1,11,11;
0003demo.sit=0,2046,1998,3,6,15,0;
0004ctrm.sit=0,2075,1998,4,1,11,12;
0005ctrm.sit=0,2368,1998,4,1,11,13;
0010csin.sit=0,1781,1998,4,1,8,31;
0020csin.sit=0,1882,1998,3,23,15,5;
1000csin.sit=0,1909,1998,3,24,20,24;
1010csin.sit=0,1802,1998,3,24,15,33;
1020csin.sit=0,1531,1998,3,24,16,40;
2000csin.sit=0,1997,1998,3,24,21,7;
2010csin.sit=0,2038,1998,3,31,13,3;
2020csin.sit=0,2383,1998,3,24,21,16;
3000csin.sit=0,2289,1998,3,25,15,44;
3010csin.sit=0,1800,1998,3,26,15,26;
3020csin.sit=0,1980,1998,3,26,12,50;
4000csin.sit=0,2290,1998,3,26,22,55;
4010csin.sit=0,1918,1998,3,27,20,25;
4020csin.sit=0,2087,1998,3,30,16,44;
cg00.uds=1,149701,1998,3,7,17,36; // 1: extract
}
```


EditHistory.cpp

```
//
//  CEditHistory
//
//  (C) Programmed by Kim, , Dec 9, 1996
//  SDS Media Lab
//  Information Technology Institue
//  unichat
//

#include "stdafx.h"
#include "UC2.h"
#include "EditHistory.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CEditHistory

CEditHistory::CEditHistory()
{
}

CEditHistory::~CEditHistory()
{
}

BEGIN_MESSAGE_MAP(CEditHistory, CEdit)    //CRichEditCtrl
//{{AFX_MSG_MAP(CEditHistory)
// NOTE - the ClassWizard will add and remove mapping macros
here.
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CEditHistory message handlers

HWND CEditHistory::GetNotepad()
{
    HWND hwndFind    = ::FindWindow("Notepad", NULL);    // "Tortoise's
Experiment"
    CTime time = CTime::GetCurrentTime();
    CString strFile(time.Format(_T("UC%y%m%d.txt")));
    if (!hwndFind)
    {
        STARTUPINFO        si;
        PROCESS_INFORMATION pi;
        ::ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        CString strCmd = _T("WRITE ") + strFile;
        BOOL bRes = ::CreateProcess("C:\\WINDOWS\\NOTEPAD.EXE",
strCmd.GetBuffer(128), NULL, NULL,
FALSE, NULL, NULL, NULL, &si, &pi);
    }
}
```

```

        if (!bRes)
            return NULL;
        ::CloseHandle(pi.hThread);
        ::WaitForInputIdle(pi.hProcess, 10000);
        ::CloseHandle(pi.hProcess);
        int nRep=0;
        while (!hwndFind && (nRep++ < 100))
        {
            ::Sleep(1000);
            TRACE("z");
            hwndFind = ::FindWindow("Notepad", NULL);    //
            "Tortoise's Experiment"
        }
    }
    // ::SendMessage(hwndFind, WM_SETTEXT, 0, (LPARAM)(LPCSTR)strFile);
    return hwndFind;
}

// 1. Find a running Notepad (or initiate a new Notepad) application.
// 2. Set the contents in the notepad (Edit control) with some texts.
void CEditHistory::SendTextToNotepad()
{
    CString strText;
    GetWindowText(strText);
    int len = strText.GetLength();
    TRACE1("CEditHistory::SendTextToNotepad - length:%d\n", len);
    if (len == 0)
        return;
    LPCSTR szText = strText;
    HWND hwndFind = GetNotepad();
    HWND hwndChild = ::GetWindow(hwndFind, GW_CHILD);
    HWND hwnd = hwndChild;
    UINT msg = WM_SETTEXT;
    WPARAM wParam = 0;
    LPARAM lParam = (LPARAM)szText;
    LRESULT res = ::SendMessage(hwnd, msg, wParam,
    lParam);
    TRACE("::SendMessage(0x%lx, msg, wParam, lParam); returned %lx\n",
        hwnd, msg, wParam, lParam, res);
}

/*
void CEditHistory::Experiment2()
{
    char szBuf[2048];
    HWND hwndFind = GetNotepad();
    HWND hwndChild = ::GetWindow(hwndFind, GW_CHILD);
    HWND hwnd = hwndChild;
    UINT msg = WM_GETTEXT;
    WPARAM wParam = sizeof(szBuf);
    LPARAM lParam = (LPARAM)szBuf;
    LRESULT res = ::SendMessage(hwnd, msg, wParam,
    lParam);
    TRACE("::SendMessage(0x%lx, msg, wParam, lParam); returned %lx\n",
        hwnd, msg, wParam, lParam, res);
    CString str = (char*)lParam;
    CRect rc;

```

EditHistory.cpp

```
        GetClientRect(&rc);  
        CClientDC dc(this);  
        dc.DrawText(str, &rc, DT_LEFT | DT_TABSTOP);  
    }  
    */
```

EditHistory.h

```
//
//  CEditHistory
//
//  (C) Programmed by Kim,
//  unichat Media Lab
//  Information Technology Institue
//  unichat
//
/////////////////////////////////////////////////////////////////
// CEditHistory window

#ifndef __EDITHISTORY_H
#define __EDITHISTORY_H

// Subclassing CEdit has the problem to halt the system for a while
// when the input data are sth. as
"....."

class CEditHistory : public CEdit    // CRichEditCtrl
{
// Construction
public:
    CEditHistory();

// Attributes
public:

// Operations
public:
    HWND          GetNotepad();
    void          SendTextToNotepad();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CEditHistory)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CEditHistory();

    // Generated message map functions
protected:
    //{{AFX_MSG(CEditHistory)
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////
#endif
```

EditSend.cpp

```
// EditSend.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "EditSend.h"
#include "MainFrm.h"
#include "UC2Doc.h"
#include "UC2Messages.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

const COLORREF crBKCOLOR = PALETTE_RGB(255,255,255);
////////////////////////////////////
// CEditSend

CEditSend::CEditSend()
{
    // m_pEditBkBrush = new CBrush(crBKCOLOR);    // 255,255,255));
}

CEditSend::~CEditSend()
{
}

BEGIN_MESSAGE_MAP(CEditSend, CEdit)
    //{AFX_MSG_MAP(CEditSend)
    ON_WM_CHAR()
    ON_WM_DESTROY()
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CEditSend message handlers

void CEditSend::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    switch (nChar)
    {
        case VK_ESCAPE:
        {
            CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
            CUC2Doc* pDoc = (CUC2Doc*)pMF->GetActiveDocument();
            ASSERT_VALID(pDoc);
            if (pDoc->IsDemo())
                pDoc->SetUC2Mode(UC2MODE_OFFLINE);
        }
        break;

        // case VK_BACK:
        //     break;

        case VK_RETURN:
    }
```

```

        if (nRepCnt == 1)
        {
            // Get current string
            CString str;
            int cl      = GetCurrentLine();      // current line (where
the caret is)
            int len = GetLine(cl, str.GetBuffer(255), 255);
            str.GetBufferSetLength(len);
            str.ReleaseBuffer();
            if (str.IsEmpty())
                return;

            ParseCommand(str);

            // Append new line char
            len = GetWindowTextLength();
            SetSel(len, len);      // starting position, ending
position
            if (cl == GetLastLine())
                ReplaceSel(_T("\r\n"));      // insert newline
        }
        break;
    }
    CEdit::OnChar(nChar, nRepCnt, nFlags);
}

void CEditSend::OnDestroy()
{
    CEdit::OnDestroy();
    // delete m_pEditBkBrush;
}

void CEditSend::ParseCommand(LPCSTR szText)
{
#ifdef _KOREAN
    static char* szMove[] =
    {
        "Àu", "ÈÄ", "ÁÂ", "¿ì"
    };
    static char* szEmotion[] =
    {
        "¿ô", "È-", "ÁÎ", "¿ì", "´Ç", "ÁÝ", "½°", "¶§"
    };
#else // English
    static char* szMove[] =
    {
        "go", "back", "left", "right"
    };
    static char* szEmotion[] =
    {
        "smile", "mad", "hello", "cry", "scratch", "pick", "special",
"hit"
    };
#endif
}

CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();

```

```

CUC2Doc* pDoc = (CUC2Doc*)pMF->GetActiveDocument();
ASSERT_VALID(pDoc);
int len = lstrlen(szText);

if ((len > 2) && (szText[0] == '/'))
{
    for (int i=0; i < (sizeof(szMove)/sizeof(szMove[0])); i++)
    {
        if ((szText[1] == szMove[i][0]) && (szText[2] ==
szMove[i][1]))
        {
            switch (i)
            {
            case 0:
            case 1:      pDoc->SendMoveCommand(CMD_MOVEF + i);
break;

            case 2:
            case 3:      pDoc->SendCommand(CMD_TURNL + i - 2);
break;

            }
            return;
        }
    }
    for (i=0; i < (sizeof(szEmotion)/sizeof(szEmotion[0])); i++)
    {
        if ((szText[1] == szEmotion[i][0]) && (szText[2] ==
szEmotion[i][1]))
        {
            pDoc->SendCommand(CMD_SMILE + i);
            return;
        }
    }
    // No commands
    pDoc->SendText(szText);
}

/*
HBRUSH CEditSend::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    switch (nCtlColor)
    {
    case CTLCOLOR_EDIT:
    case CTLCOLOR_MSGBOX:
        pDC->SetTextColor(PALETTE_RGB(0,0,0));      // Yellow
        pDC->SetBkColor(crBKGOLOR);
        return (HBRUSH)(m_pEditBkBrush->GetSafeHandle());
    default:
        return CEdit::OnCtlColor(pDC, pWnd, nCtlColor);
    }
}
*/

```

EditSend.h

```
#if !defined(AFX_EDITSEND_H_E4B02724_B4CD_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_EDITSEND_H_E4B02724_B4CD_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// EditSend.h : header file
//

/////////////////////////////////////////////////////////////////
// CEditSend window

class CEditSend : public CEdit
{
// Construction
public:
    CEditSend();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CEditSend)
    //}}AFX_VIRTUAL

// Implementation
protected:
    int GetCurrentLine() const { return LineFromChar(LineIndex(-
1)); } // index from 0
    int GetLastLine() const { return (GetLineCount() -
1); }
    void ParseCommand(LPCSTR szText);
    // CBrush* m_pEditBkBrush;

public:
    virtual ~CEditSend();

    // Generated message map functions
protected:
    //{{AFX_MSG(CEditSend)
    afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
    afx_msg void OnDestroy();
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
```


EditSend.h

```
#endif //  
!defined(AFX_EDITSEND_H__E4B02724_B4CD_11D1_80E2_080009B9F339__INCLUDED_)
```

```

//*****
// FlatToolBar.cpp
// (c) 1997, Roger Onslow
//=====
//      (C) Programmed by KEN Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT
//=====

#include "stdafx.h"
#include "FlatToolBar.h"

#ifdef _DEBUG
#undef THIS_FILE
#define new DEBUG_NEW
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

BEGIN_MESSAGE_MAP(CFlatToolBar, CToolBar)
//{{AFX_MSG_MAP(CFlatToolBar)
ON_WM_WINDOWPOSCHANGING()
ON_WM_PAINT()
ON_WM_NCPAINT()
ON_WM_NCCALCSIZE()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

IMPLEMENT_DYNAMIC(CFlatToolBar, CToolBar)

void CFlatToolBar::SetFlatLookStyle()
{
    // Set flat style (transparent)
    ModifyStyle(0, TBSTYLE_FLAT);
    // others are...
    // #define TBSTYLE_TOOLTIPS          0x0100
    // #define TBSTYLE_WRAPABLE          0x0200
    // #define TBSTYLE_ALTDRAW          0x0400
    // #define TBSTYLE_FLAT              0x0800
    // #define TBSTYLE_LIST              0x1000
}

void CFlatToolBar::RepaintBackground()
{
    CRect rc;
    GetWindowRect(&rc); // get rect for toolbar
    CWnd* pParent = GetParent(); // get parent windows
    pParent->ScreenToClient(&rc); // convert to parent coords
    pParent->InvalidateRect(&rc); // paint rectangle underneath
}

void CFlatToolBar::DrawSeparators()
{
    CClientDC dc(this); // get a dc for the client area
    DrawSeparators(&dc); // draw the separators on it
}

```

```

void CFlatToolBar::DrawSeparators(CClientDC* pDC)
{
    // horizontal vs vertical
    bool ishorz = (m_dwStyle & CBRS_ORIENT_HORZ) != 0;
    // get number of buttons
    int nIndexMax = (int)DefWindowProc(TB_BUTTONCOUNT, 0, 0);
    int nIndex;
    // try each button
    for (nIndex = 0; nIndex < nIndexMax; nIndex++)
    {
        UINT dwStyle = GetButtonStyle(nIndex);
        UINT wStyle = LOWORD(dwStyle);
        // if it is a separator
        if (wStyle == TBBS_SEPARATOR)
        {
            // get it's rectangle and width
            CRect rect;
            GetItemRect(nIndex, rect);
            // if small enough to be a true separator
            int w = rect.Width();
            if (w <= 8)
            {
                if (ishorz)
                {
                    // draw the separator bar in the middle
                    CRect rectbar = rect;
                    int x = (rectbar.left+rectbar.right)/2;
                    rectbar.left = x-1; rectbar.right = x+1;
                    pDC->Draw3dRect(rectbar, ::GetSysColor(COLOR_3DSHADOW),
                        ::GetSysColor(COLOR_3DHILIGHT));
                }
                else
                {
                    // draw the separator bar in the middle
                    CRect rectbar = rect;
                    rectbar.left = rectbar.left - m_sizeButton.cx;
                    rectbar.right = rectbar.left + m_sizeButton.cx;
                    rectbar.top = rectbar.bottom+1;
                    rectbar.bottom = rectbar.top+3;
                    int y = (rectbar.top+rectbar.bottom)/2;
                    rectbar.top = y-1; rectbar.bottom = y+1;
                    pDC->Draw3dRect(rectbar, ::GetSysColor(COLOR_3DSHADOW),
                        ::GetSysColor(COLOR_3DHILIGHT));
                }
            }
        }
    }
}

void CFlatToolBar::DrawGripper(CWindowDC *pDC, CRect& rectWindow)
{
    CRect gripper = rectWindow;
    gripper.DeflateRect(1,1);
    if (m_dwStyle & CBRS_FLOATING)
    {

```

```

        // no grippers
    }
    else if (m_dwStyle & CBR5_ORIENT_HORZ)
    {
        // gripper at left
        gripper.right = gripper.left+3;
        pDC->Draw3dRect(gripper, ::GetSysColor(COLOR_3DHIGHLIGHT),
            ::GetSysColor(COLOR_3DSHADOW));
        gripper.OffsetRect(+4, 0);
        pDC->Draw3dRect(gripper, ::GetSysColor(COLOR_3DHIGHLIGHT),
            ::GetSysColor(COLOR_3DSHADOW));
        rectWindow.left += 8;
    }
    else
    {
        // gripper at top
        gripper.bottom = gripper.top+3;
        pDC->Draw3dRect(gripper, ::GetSysColor(COLOR_3DHIGHLIGHT),
            ::GetSysColor(COLOR_3DSHADOW));
        gripper.OffsetRect(0, +4);
        pDC->Draw3dRect(gripper, ::GetSysColor(COLOR_3DHIGHLIGHT),
            ::GetSysColor(COLOR_3DSHADOW));
        rectWindow.top += 8;
    }
}

void CFlatToolBar::EraseNonClient()
{
    // get window DC that is clipped to the non-client area
    CWindowDC dc(this);
    CRect rectClient;
    GetClientRect(rectClient);
    CRect rectWindow;
    GetWindowRect(rectWindow);
    ScreenToClient(rectWindow);
    rectClient.OffsetRect(-rectWindow.left, -rectWindow.top);
    dc.ExcludeClipRect(rectClient);

    // draw borders in non-client area
    rectWindow.OffsetRect(-rectWindow.left, -rectWindow.top);
    DrawBorders(&dc, rectWindow);

    // erase parts not drawn
    dc.IntersectClipRect(rectWindow);
    SendMessage(WM_ERASEBKGD, (WPARAM)dc.m_hDC);

    DrawGripper(&dc, rectWindow); // <-- my addition to draw gripper
}

void CFlatToolBar::OnUpdateCmdUI(CFrameWnd* pTarget, BOOL bDisableIfNoHndler)
{
    static CUIntArray styles;
    // save styles
    int nIndexMax = (int)DefWindowProc(TB_BUTTONCOUNT, 0, 0);
    int nIndex;
    for (nIndex = 0; nIndex < nIndexMax; nIndex++)
    {

```

```

        UINT dwStyle = GetButtonStyle(nIndex);
        styles.SetAtGrow(nIndex,dwStyle);
    }
    // default processing
    CToolBar::OnUpdateCmdUI(pTarget,bDisableIfNoHndler);
    // check for changes to style (buttons pressed/released)
    for (nIndex = 0; nIndex < nIndexMax; nIndex++)
    {
        UINT dwStyle = GetButtonStyle(nIndex);
        if (styles[nIndex] != dwStyle)
        {
            RepaintBackground();    // need to take care of button
background
            Invalidate();           // repaint toolbar (not just this
button)
            break;
        }
    }
}

void CFlatToolBar::OnWindowPosChanging(LPWINDOWPOS lppw)
{
    // default processing
    CToolBar::OnWindowPosChanging(lppw);
    // repaint background if size or move
    if (!(lppw->flags & SWP_NOMOVE) || !(lppw->flags & SWP_NOSIZE))
    {
        // if moved:
        RepaintBackground();
        PostMessage(WM_NCPAINT);
    }
}

void CFlatToolBar:: OnPaint()
{
    // standard toolbar
    CToolBar::OnPaint();
    // plus separators
    DrawSeparators();
}

void CFlatToolBar:: OnNcPaint()
{
    EraseNonClient();
}

void CFlatToolBar::OnNcCalcSize(BOOL bCalcValidRects, NCCALCSIZE_PARAMS*
lpncsp)
{
    CToolBar::OnNcCalcSize(bCalcValidRects,lpncsp);
    // adjust non-client area for gripper at left or top
    if (m_dwStyle & CBRS_FLOATING)
    {
        // no grippers
    }
    else if (m_dwStyle & CBRS_ORIENT_HORZ)
    {
        lpncsp->rgrc[0].left += 2;
    }
}

```

[illegible]

FlatToolBar.h

```
// FlatToolBar.h

#ifndef __FLATTOOLBAR_H
#define __FLATTOOLBAR_H

class CFlatToolBar : public CToolBar
{
DECLARE_DYNAMIC(CFlatToolBar);
public:
    void SetFlatLookStyle();
    void RepaintBackground();
    void DrawSeparators();
    void DrawSeparators(CClientDC* pDC);
    void EraseNonClient();
    void DrawGripper(CWindowDC *pDC, CRect& rectWindow);
protected:
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFlatToolBar)
    virtual void OnUpdateCmdUI(CFrameWnd* pTarget, BOOL
bDisableIfNoHndler);
    //}}AFX_VIRTUAL
    // Message Handlers
protected:
    //{{AFX_MSG(CFlatToolBar)
    afx_msg void OnWindowPosChanging(LPWINDOWPOS lpWndPos);
    afx_msg void OnPaint();
    afx_msg void OnNcPaint();
    afx_msg void OnNcCalcSize( BOOL bCalcValidRects, NCCALCSIZE_PARAMS*
lpncsp );
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP();
};

#endif // __FLATTOOLBAR_H
```

httpUtility.cpp

```
// httpUtility.cpp -- contains stuff used by more than one thread

#include <stdafx.h>
#include <afxmt.h>

#include "blocksock.h"
#include "httpUtility.h"

volatile int g_nConnection = 0; // connection number
CString g_strServerName = "localhost"; // used by both winsock and wininet
CString g_strServerIP;
volatile UINT g_nPort = 80;
CString g_strReq = "/custom";

HWND g_hMainWnd = 0;
char g_pchStatus[25] = "";
CCriticalSection g_csStatus;

CCallbackInternetSession::CCallbackInternetSession( LPCTSTR pstrAgent, DWORD
dwContext,
    DWORD dwAccessType, LPCTSTR pstrProxyName, LPCTSTR pstrProxyBypass,
    DWORD dwFlags) :
    CInternetSession(pstrAgent, dwContext, dwAccessType, pstrProxyName,
pstrProxyBypass, dwFlags)
{
    EnableStatusCallback();
}

void CCallbackInternetSession::OnStatusCallback(DWORD dwContext, DWORD
dwInternalStatus,
    LPVOID lpvStatusInformation, DWORD dwStatusInformationLength)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    int errors[] = {10, 11, 20, 21, 30, 31, 40, 41, 42, 43, 50, 51, 60, 70,
100, 110, 0};
    char* text[] = {
        "Resolving name",
        "Name resolved",
        "Connecting to server",
        "Connected to server",
        "Sending request",
        "Request sent",
        "Receiving response",
        "Response received",
        "Ctl response received",
        "Prefetch",
        "Closing connection",
        "Connection closed",
        "Handle created",
        "Handle closing",
        "Request complete",
        "Redirect",
        "Unknown" };
    int n;
    /* // demonstrates request cancellation
```



```

        if(dwInternalStatus == INTERNET_STATUS_REQUEST_SENT) {
            AfxThrowInternetException(dwContext, 999);
        }
    */
    for(n = 0; errors[n] != 0; n++) {
        if(errors[n] == (int) dwInternalStatus) break;
    }
    g_csStatus.Lock();
    strcpy(g_pchStatus, text[n]);
    if(dwInternalStatus == INTERNET_STATUS_RESOLVING_NAME ||
        dwInternalStatus == INTERNET_STATUS_NAME_RESOLVED) {
        strcat(g_pchStatus, "-");
        strcat(g_pchStatus, (char*) lpvStatusInformation);
    }
    TRACE("WININET STATUS: %s\n", g_pchStatus);
    g_csStatus.Unlock();
    // frame doesn't need a handler -- message triggers OnIdle, which
updates status bar
    ::PostMessage(g_hMainWnd, WM_CALLBACK, 0, 0);
}

void LogInternetException(LPVOID pParam, CInternetException* pe)
{
    // pParam holds the HWND for the destination window (in another thread)
    CString strGmt = CTime::GetCurrentTime().FormatGmt("%m/%d/%y %H:%M:%
GMT");
    char text1[300], text2[100];
    wsprintf(text1, "CLIENT ERROR: WinInet error #%d -- %s\r\n    ",
        pe->m_dwError, (const char*) strGmt);
    pe->GetErrorMessage(text2, 99);
    strcat(text1, text2);
    if(pe->m_dwError == 12152) {
        strcat(text1, " URL not found?\r\n");
    }
    ::SendMessage((HWND) pParam, EM_SETSEL, (WPARAM) 65534, 65535);
    ::SendMessage((HWND) pParam, EM_REPLACESEL, (WPARAM) 0, (LPARAM)
text1);
}

```

```
// httpUtility.h

#define WM_CALLBACK WM_USER + 5
extern volatile int g_nConnection;
extern CString g_strServerName; // used by both winsock and wininet code
extern CString g_strServerIP; // used by both winsock and wininet code
extern CString g_strReq;
extern char g_pchStatus[];
extern HWND g_hMainWnd;
extern CCriticalSection g_csStatus;
extern CString g_strIPClient;
extern volatile UINT g_nPort;
extern CString g_strProxy;
extern BOOL g_bUseProxy;
extern volatile BOOL g_bListening;
// extern CString g_strDirect;
extern CString g_strIPServer;
extern volatile UINT g_nPortServer;
extern CString g_strURL;
//extern CString g_strDefault;

extern UINT ClientUrlThreadProc(LPVOID pParam);
extern UINT ServerThreadProc(LPVOID pParam);
extern UINT ClientWinInetThreadProc(LPVOID pParam);
extern UINT ClientSocketThreadProc(LPVOID pParam);

extern void LogInternetException(LPVOID pParam, CInternetException* pe);

class CCallbackInternetSession : public CInternetSession
{
public:
    CCallbackInternetSession( LPCTSTR pstrAgent = NULL, DWORD dwContext =
1,
        DWORD dwAccessType = PRE_CONFIG_INTERNET_ACCESS, LPCTSTR
pstrProxyName = NULL,
        LPCTSTR pstrProxyBypass = NULL, DWORD dwFlags = 0 );
protected:
    virtual void OnStatusCallback(DWORD dwContext, DWORD dwInternalStatus,
        LPVOID lpvStatusInformation, DWORD dwStatusInformationLength);
};
```

InputDialog.cpp

```
// InputDlg.cpp : implementation file
//

#include "stdafx.h"
#include "BMC.h"
#include "InputDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////

// CInputDialog dialog

CInputDialog::CInputDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CInputDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(CInputDialog)
    m_strData = _T("");
    //}}AFX_DATA_INIT
}

void CInputDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CInputDialog)
    DDX_Text(pDX, IDC_EDIT_STRING, m_strData);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CInputDialog, CDialog)
    //{{AFX_MSG_MAP(CInputDialog)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////

// CInputDialog message handlers
```

InputDlg.h

```
#if !defined(AFX_INPUTDLG_H__2F1ADE63_AD1A_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_INPUTDLG_H__2F1ADE63_AD1A_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// InputDlg.h : header file
//

/////////////////////////////////////////////////////////////////
// CInputDlg dialog

class CInputDlg : public CDialog
{
// Construction
public:
    CInputDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CInputDlg)
    enum { IDD = IDD_INPUT_DLG };
    CString        m_strData;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CInputDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CInputDlg)
    // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#endif(AFX_INPUTDLG_H__2F1ADE63_AD1A_11D1_80E2_080009B9F339__INCLUDED_)
```

InputIntDlg.cpp

```
// InputIntDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "InputIntDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CInputIntDlg dialog

CInputIntDlg::CInputIntDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CInputIntDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CInputIntDlg)
    // m_nVal = 0;
    //}}AFX_DATA_INIT
}

void CInputIntDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CInputIntDlg)
    DDX_Text(pDX, IDC_EDIT_INT, m_nVal);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CInputIntDlg, CDialog)
    //{{AFX_MSG_MAP(CInputIntDlg)
    ON_BN_CLICKED(IDC_BTN_DEFAULT, OnBtnDefault)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// CInputIntDlg message handlers

void CInputIntDlg::OnBtnDefault()
{
    m_nVal = m_nDefaultVal;
    UpdateData(FALSE);    // initialize dialog box
}

BOOL CInputIntDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    SetWindowText(m_strTitle);
    UpdateData(FALSE);    // initialize dialog box
}
```

InputIntDlg.cpp

```
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
```

InputIntDlg.h

```

#if
!defined(AFX_INPUTINTDLG_H__E57FAA23_27A2_11D2_80E2_080009B9F339__INCLUDED_)
#define AFX_INPUTINTDLG_H__E57FAA23_27A2_11D2_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// InputIntDlg.h : header file
//

/////////////////////////////////////////////////////////////////
// CInputIntDlg dialog

class CInputIntDlg : public CDialog
{
// Construction
public:
    CInputIntDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{{AFX_DATA(CInputIntDlg)
    enum { IDD = IDD_INPUT_INT };
    int         m_nVal;
    //}}AFX_DATA

    void SetDefaultVal(const int nDV) { m_nDefaultVal = nDV; }
    void SetTitle(const CString& str) { m_strTitle = str; }

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CInputIntDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    int         m_nDefaultVal;
    CString     m_strTitle;

    // Generated message map functions
    //{{AFX_MSG(CInputIntDlg)
    afx_msg void OnBtnDefault();
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_INPUTINTDLG_H__E57FAA23_27A2_11D2_80E2_080009B9F339__INCLUDED_)

```

InputPassword.cpp

```
// InputPassword.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "InputPassword.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////

// CInputPassword dialog

CInputPassword::CInputPassword(CWnd* pParent /*=NULL*/)
    : CDialog(CInputPassword::IDD, pParent)
{
    //{{AFX_DATA_INIT(CInputPassword)
    m_strPassword = _T("");
    //}}AFX_DATA_INIT
}

void CInputPassword::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CInputPassword)
    DDX_Text(pDX, IDC_EDIT_PASSWORD, m_strPassword);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CInputPassword, CDialog)
    //{{AFX_MSG_MAP(CInputPassword)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CInputPassword message handlers
```


InputPassword.h

```
#if
!defined(AFX_INPUTPASSWORD_H__A1BFE564_C321_11D1_80E2_080009B9F339__INCLUDED_
)
#define AFX_INPUTPASSWORD_H__A1BFE564_C321_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// InputPassword.h : header file
//

////////////////////////////////////
// CInputPassword dialog

class CInputPassword : public CDialog
{
// Construction
public:
    CInputPassword(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CInputPassword)
    enum { IDD = IDD_INPUT_PASSWORD };
    CString        m_strPassword;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CInputPassword)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CInputPassword)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_INPUTPASSWORD_H__A1BFE564_C321_11D1_80E2_080009B9F339__INCLUDED_
) \
```

JunAsyncMF.cpp

```
// JunAsyncMF.cpp : implementation file
//

#include "stdafx.h"
#include "Test.h"
#include "JunAsyncMF.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CJunAsyncMF

CJunAsyncMF::CJunAsyncMF()
{
}

CJunAsyncMF::~CJunAsyncMF()
{
}

// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CJunAsyncMF, CAsyncMonikerFile)
    //{AFX_MSG_MAP(CJunAsyncMF)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

//////////////////////////////////////
// CJunAsyncMF member functions

void CJunAsyncMF::OnDataAvailable(DWORD dwSize, DWORD bscfFlag)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunAsyncMF::OnDataAvailable\n");
    TRACE("dwSize = %ld , bscfFlag = %ld\n", dwSize, bscfFlag);

    CAsyncMonikerFile::OnDataAvailable(dwSize, bscfFlag);
}

void CJunAsyncMF::OnStartBinding(DWORD grfBSCOPTION)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunAsyncMF::OnStartBinding\n");

    //CAsyncMonikerFile::OnStartBinding(grfBSCOPTION);
}

void CJunAsyncMF::OnStopBinding(HRESULT hresult, LPCTSTR szError)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunAsyncMF::OnStopBinding\n");
}
```

```
        TRACE("hresult = %d\n", hresult);
        CAsyncMonikerFile::OnStopBinding(hresult, szError);
    }

void CJunAsyncMF::OnProgress(ULONG ulProgress, ULONG ulProgressMax,
    ULONG ulStatusCode, LPCTSTR szStatusText)
{
    TRACE("CJunAsyncMF::OnProgress\n");
    TRACE("ulProgress : %ld\n", ulProgress);
    TRACE("ulProgressMax : %ld\n", ulProgressMax);
    TRACE("ulStatusCode : %ld\n", ulStatusCode);
    TRACE("szStatusText : %s\n", szStatusText);

    CAsyncMonikerFile::OnProgress(ulProgress, ulProgressMax, ulStatusCode,
    szStatusText);
}
```

JunAsyncMF.h

```
#if
!defined(AFX_JUNASYNCFM_H__7F6B9704_B810_11D1_9169_444553540001__INCLUDED_)
#define AFX_JUNASYNCFM_H__7F6B9704_B810_11D1_9169_444553540001__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// JunAsyncMF.h : header file
//

////////////////////////////////////
// CJunAsyncMF command target

class CJunAsyncMF : public CAsyncMonikerFile
{
// Attributes
public:

// Operations
public:
    CJunAsyncMF();
    virtual ~CJunAsyncMF();

// Overrides
public:
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CJunAsyncMF)
protected:
    virtual void OnDataAvailable(DWORD dwSize, DWORD bscfFlag);
    virtual void OnStartBinding(DWORD grfBSCOPTION);
    virtual void OnStopBinding(HRESULT hresult, LPCTSTR szError);
    virtual void OnProgress(ULONG ulProgress, ULONG ulProgressMax,
        ULONG ulStatusCode, LPCTSTR szStatusText);
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(CJunAsyncMF)
        // NOTE - the ClassWizard will add and remove member functions
here.
    //}}AFX_MSG

// Implementation
protected:
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_JUNASYNCFM_H__7F6B9704_B810_11D1_9169_444553540001__INCLUDED_)
```

```

// JunAsyncMF.cpp : implementation file
//

#include "stdafx.h"
#include "Test.h"
#include "JunAsyncMF.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////

// CJunAsyncMF

CJunAsyncMF::CJunAsyncMF()
{
}

CJunAsyncMF::~CJunAsyncMF()
{
}

// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CJunAsyncMF, CAsyncMonikerFile)
    //{AFX_MSG_MAP(CJunAsyncMF)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

////////////////////////////////////
// CJunAsyncMF member functions

void CJunAsyncMF::OnDataAvailable(DWORD dwSize, DWORD bscfFlag)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunAsyncMF::OnDataAvailable\n");
    TRACE("dwSize = %ld , bscfFlag = %ld\n", dwSize, bscfFlag);

    CAsyncMonikerFile::OnDataAvailable(dwSize, bscfFlag);
}

void CJunAsyncMF::OnStartBinding(DWORD grfBSCOPTION)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunAsyncMF::OnStartBinding\n");

    //CAsyncMonikerFile::OnStartBinding(grfBSCOPTION);
}

void CJunAsyncMF::OnStopBinding(HRESULT hresult, LPCTSTR szError)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunAsyncMF::OnStopBinding\n");
}

```

```
        TRACE("hresult = %d\n", hresult);
        CAsyncMonikerFile::OnStopBinding(hresult, szError);
    }

void CJunAsyncMF::OnProgress(ULONG ulProgress, ULONG ulProgressMax,
    ULONG ulStatusCode, LPCTSTR szStatusText)
{
    TRACE("CJunAsyncMF::OnProgress\n");
    TRACE("ulProgress : %ld\n", ulProgress);
    TRACE("ulProgressMax : %ld\n", ulProgressMax);
    TRACE("ulStatusCode : %ld\n", ulStatusCode);
    TRACE("szStatusText : %s\n", szStatusText);

    CAsyncMonikerFile::OnProgress(ulProgress, ulProgressMax, ulStatusCode,
szStatusText);
}
```

JunDataPathProp.cpp

```
// JunDataPathProp.cpp : implementation file
//

#include "stdafx.h"
#include "Test.h"
#include "JunDataPathProp.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define WM_STOPBINDING WM_USER+1
////////////////////////////////////

CJunDataPathProp::CJunDataPathProp()
{
}

CJunDataPathProp::~CJunDataPathProp()
{
}

// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CJunDataPathProp, CDataPathProperty)
    //{AFX_MSG_MAP(CJunDataPathProp)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

////////////////////////////////////
// CJunDataPathProp member functions

void CJunDataPathProp::OnStartBinding()
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunDataPathProp::OnStartBinding\n\n");
    CDataPathProperty::OnStartBinding();
}

void CJunDataPathProp::OnStopBinding(HRESULT hresult, LPCTSTR szError)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CJunDataPathProp::OnStopBinding\n\n");

    AfxGetMainWnd()->PostMessage(WM_STOPBINDING, 0, 0);

    CDataPathProperty::OnStopBinding(hresult, szError);
}

void CJunDataPathProp::OnDataAvailable(DWORD dwSize, DWORD bscfFlag)
{
    // TODO: Add your specialized code here and/or call the base class
}
```

```

TRACE("CJunDataPathProp::OnDataAvailable\n");
TRACE("dwSize : %ld ", dwSize);
TRACE(" bscfFlag : %ld\n", bscfFlag);

if( (bscfFlag & BSCF_FIRSTDATANOTIFICATION) != 0)
{
    TRACE("BSCF_FIRSTDATANOTIFICATION\n\n");
}

if( (bscfFlag & BSCF_INTERMEDIATEDATANOTIFICATION) != 0)
{
    TRACE("BSCF_INTERMEDIATEDATANOTIFICATION\n\n");
}

if( (bscfFlag & BSCF_LASTDATANOTIFICATION) != 0)
{
    TRACE("BSCF_LASTDATANOTIFICATION\n\n");
}

if( (bscfFlag & BSCF_DATAFULLYAVAILABLE) != 0)
{
    TRACE("BSCF_DATAFULLYAVAILABLE\n\n");
}

if( (bscfFlag & BSCF_AVAILABLEDATASIZEUNKNOWN) != 0)
{
    TRACE("BSCF_AVAILABLEDATASIZEUNKNOWN\n\n");
}

CDataPathProperty::OnDataAvailable(dwSize, bscfFlag);
}

void CJunDataPathProp::OnProgress(ULONG ulProgress, ULONG ulProgressMax,
    ULONG ulStatusCode, LPCTSTR szStatusText)
{
    TRACE("CJunDataPathProp::OnProgress\n");
    TRACE("ulProgress : %ld\n", ulProgress);
    TRACE("ulProgressMax : %ld\n", ulProgressMax);
    TRACE("ulStatusCode : %ld\n", ulStatusCode);

    if(ulStatusCode == BINDSTATUS_FINDINGRESOURCE)
    {
        TRACE("ulStatusCode : BINDSTATUS_FINDINGRESOURCE\n");
    }
    else if(ulStatusCode == BINDSTATUS_MIMETYPEAVAILABLE)
    {
        TRACE("ulStatusCode : BINDSTATUS_MIMETYPEAVAILABLE\n");
    }
    else if(ulStatusCode == BINDSTATUS_CONNECTING)
    {
        TRACE("ulStatusCode : BINDSTATUS_CONNECTING\n");
    }
    else if(ulStatusCode == BINDSTATUS_SENDINGREQUEST)
    {
        TRACE("ulStatusCode : BINDSTATUS_SENDINGREQUEST\n");
    }
    else if(ulStatusCode == BINDSTATUS_REDIRECTING)

```



```

{
    TRACE("ulStatusCode : BINDSTATUS_REDIRECTING\n");
}
else if (ulStatusCode == BINDSTATUS_USINGCACHEDCOPY)
{
    TRACE("ulStatusCode : BINDSTATUS_USINGCACHEDCOPY\n");
}
else if (ulStatusCode == BINDSTATUS_BEGINDOWNLOADDATA)
{
    TRACE("ulStatusCode : BINDSTATUS_BEGINDOWNLOADDATA\n");
}
else if (ulStatusCode == BINDSTATUS_DOWNLOADINGDATA)
{
    TRACE("ulStatusCode : BINDSTATUS_DOWNLOADINGDATA\n");
}
else if (ulStatusCode == BINDSTATUS_ENDDOWNLOADDATA)
{
    TRACE("ulStatusCode : BINDSTATUS_ENDDOWNLOADDATA\n");
}
else if (ulStatusCode == BINDSTATUS_CACHEFILENAMEAVAILABLE)
{
    TRACE("ulStatusCode : BINDSTATUS_CACHEFILENAMEAVAILABLE\n");
}

TRACE("szStatusText : %s\n\n", szStatusText);

CDataPathProperty::OnProgress (ulProgress, ulProgressMax, ulStatusCode,
szStatusText);
}

```

JunDataPathProp.h

```

#if
!defined(AFX_JUNDATAPATHPROP_H__5959C5C2_B813_11D1_9169_444553540001__INCLUDE
D_)
#define
AFX_JUNDATAPATHPROP_H__5959C5C2_B813_11D1_9169_444553540001__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// JunDataPathProp.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CJunDataPathProp command target

class CJunDataPathProp : public CDataPathProperty
{
// Attributes
public:

// Operations
public:
    CJunDataPathProp();
    virtual ~CJunDataPathProp();

// Overrides
public:
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CJunDataPathProp)
protected:
    virtual void OnStartBinding();
    virtual void OnStopBinding(HRESULT hresult, LPCTSTR szError);
    virtual void OnDataAvailable(DWORD dwSize, DWORD bscfFlag);
    virtual void OnProgress(ULONG ulProgress, ULONG ulProgressMax,
        ULONG ulStatusCode, LPCTSTR szStatusText);
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(CJunDataPathProp)
        // NOTE - the ClassWizard will add and remove member functions
here.
    //}}AFX_MSG

// Implementation
protected:
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

```

JunDataPathProp.h

```
#endif //  
!defined(AFX_JUNDATAPATHPROP_H__5959C5C2_B813_11D1_9169_444553540001__INCLUDE  
D_)
```

LoginDlg.cpp

```
// LoginDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "LoginDlg.h"
#include "ResMan.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"
#include "UC2CS.h"
#include "InputPassword.h"
#include "Splash.h"

#include "MainFrm.h"
#include "UC2Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

extern int  GetClientVersion();      // UC2Doc.cpp

const int MSPT_TIMER = 400;
#define ArrayCount(a)  (sizeof(a)/sizeof(a[0]))

#ifdef _MALL
LPCTSTR BMP_LOGIN_BACK          = "MLoginBk.bmp";
//LPCTSTR BMP_LOGIN_ANI          = "MLgnAni.bmp";

LPCTSTR BMP_LOGIN_BTN_OK        = "MBtnOK.bmp";
LPCTSTR BMP_LOGIN_BTN_NO        = "MBtnNo.bmp";

//LPCTSTR BMP_LOGIN_BTN_LAN      = "MBtnLAN.bmp";
//LPCTSTR BMP_LOGIN_BTN_MODEM    = "MBtnMdm.bmp";
//const CPoint  PTLT_ANI(85, 98);
//const CPoint  PTLT_LM[2] = {CPoint(276, 219), CPoint(276, 171)};    //
LAN or Modem
//const CRect   RECT_NICKNAME(139, 209, 246, 229);
//const CRect   RECT_PASSWORD(139, 232, 246, 251);
const CRect    RECT_NICKNAME(111, 186, 218, 206);
const CRect    RECT_PASSWORD(111, 212, 218, 232);
const CRect    RECT_MESSAGE(50, 239, 243, 259);
const CRect    RECT_SECONDS(306, 145, 374, 165);
const CRect    RECT_HOST(62, 14, 315, 29+100);
//const int BEHAVIOR0[] = {0,0,1,0,0,0,0,1,0,0};
//const int BEHAVIOR1[] = {0,2,3,4,6,3,4,6,3,4,6};
//const int BEHAVIOR2[] = {0,4,7,4,7,4,7};
#else
//LPCTSTR BMP_LOGIN_BACK          = "U2Login|LoginBk.bmp";
LPCTSTR BMP_LOGIN_BACK          = "login_screen.bmp";

//LPCTSTR BMP_LOGIN_OVAL          = "U2Login|LoginOva.bmp";
//LPCTSTR BMP_LOGIN_ANI          = "U2Login|LoginAni.bmp";
```

LoginDlg.cpp

```
//LPCTSTR BMP_LOGIN_BTN_OK      = "U2Login|BtnOK.bmp";
//LPCTSTR BMP_LOGIN_BTN_NO      = "U2Login|BtnNo.bmp";

LPCTSTR BMP_LOGIN_BTN_OK      = "login_ok.bmp";
LPCTSTR BMP_LOGIN_BTN_NO      = "login_cancel.bmp";

//LPCTSTR BMP_LOGIN_BTN_LAN     = "U2Login|BtnLAN.bmp";
//LPCTSTR BMP_LOGIN_BTN_MODEM   = "U2Login|BtnMODEM.bmp";
//const CPoint      PTLT_OVAL(91, 12);
//const CPoint      PTLT_ANI(161, 40);
//const CPoint      PTLT_LM[2] = {CPoint(120, 207), CPoint(207, 207)};    //
LAN or Modem

//const CRect      RECT_NICKNAME(158, 159, 264, 180);
//const CRect      RECT_PASSWORD(158, 189, 264, 207);
const CRect      RECT_NICKNAME(139, 137, 279, 155);
const CRect      RECT_PASSWORD(139, 162, 279, 180);

//const CRect      RECT_MESSAGE(90, 258, 340, 285);
const CRect      RECT_MESSAGE(20, 115, 100, 155);
const CRect      RECT_SECONDS(125, 115, 150, 135);
const CRect      RECT_HOST(105, 7, 325, 22+100);

const int BEHAVIOR0[] = {0,0,1,0,0,0,0,1,0,1,0};
const int BEHAVIOR1[] = {0,2,3,3,3,3,3,2,3};
const int BEHAVIOR2[] = {4,5,6, 7,8,9,10,11, 7,8,9,10,11, 7,8,9,10,11};
const int* ANI_BEH[] = {BEHAVIOR0, BEHAVIOR1, BEHAVIOR2};
const int ANI_BEH_COUNT[] =
{
    ArrayCount(BEHAVIOR0),
    ArrayCount(BEHAVIOR1),
    ArrayCount(BEHAVIOR2)
};
#endif

////////////////////////////////////
// CLoginDlg message handlers
BEGIN_MESSAGE_MAP(CLoginDlg, CDialog)
//{{AFX_MSG_MAP(CLoginDlg)
    ON_WM_ERASEBKGD()
    ON_WM_QUERYNEWPALETTE()
    ON_WM_PALETTECHANGED()
    ON_WM_SIZE()
    ON_WM_TIMER()
    ON_WM_DESTROY()
    ON_WM_SETCURSOR()
    ON_WM_LBUTTONDOWN()
    ON_EN_SETFOCUS(IDC_EDIT_NICKNAME, OnSetfocusEditNickname)
    ON_EN_SETFOCUS(IDC_EDIT_PASSWORD, OnSetfocusEditPassword)
    ON_WM_LBUTTONDBLCLK()
    ON_WM_CTLCOLOR()
    ON_EN_UPDATE(IDC_EDIT_NICKNAME, OnUpdateEditNickname)
    ON_EN_UPDATE(IDC_EDIT_PASSWORD, OnUpdateEditPassword)
    ON_EN_KILLFOCUS(IDC_EDIT_NICKNAME, OnKillfocusEditNickname)
    ON_EN_KILLFOCUS(IDC_EDIT_PASSWORD, OnKillfocusEditPassword)
}}
```

```

    ON_CBN_EDITUPDATE(IDC_CB_HOST, OnEditupdateCbHost)
    ON_CBN_KILLFOCUS(IDC_CB_HOST, OnKillfocusCbHost)
    ON_WM_PAINT()
    ON_WM_NCHITTEST()
    //}}AFX_MSG_MAP
    ON_MESSAGE(CMD_CONNECT_CONNECTING, OnConnectConnecting)
    ON_MESSAGE(CMD_CONNECT_LOGIN, OnConnectLogin)
    ON_MESSAGE(CMD_CONNECT_BACKUPID, OnConnectBackupID)
    ON_MESSAGE(CMD_CONNECT_FAILURE, OnConnectFailure)
END_MESSAGE_MAP()

////////////////////////////////////
// CLoginDlg dialog

CLocalDlg::CLocalDlg(CUC2Socket* pSocket, CWnd* pParent /*=NULL*/)
: CDialog(CLocalDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CLocalDlg)
    m_strHost = _T("");
    m_strPassword = _T("");
    m_strSec = _T("");
    //}}AFX_DATA_INIT
    m_bConnecting = FALSE;
    m_uiTimer = 0;
    m_pConnThread = NULL;
    m_hConnThread = NULL;
    m_pSocket = pSocket;

    ASSERT(m_pSocket);
    m_strMessage.LoadString(IDS_LOGIN_BEGIN);
    m_bFirstFocus = TRUE;
#ifdef _MALL
    m_bPause = TRUE;
#endif
    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + BMP_LOGIN_BACK);
    m_pDIBBack = new CDIB;
    if (!m_pDIBBack->Load(strFile))
    {
        delete m_pDIBBack;
        m_pDIBBack = NULL;
        return;
    }

    // Create the palette from the DIB.
    m_pPal = new CDIBPal;
    ASSERT(m_pPal);
    if (!m_pPal->Create(m_pDIBBack))
    {
        AfxMessageBox("Failed to create palette from DIB file");
        delete m_pPal;
        m_pPal = NULL;
    }

#ifdef _MALL
    /*
    strFile = strPath + BMP_LOGIN_OVAL;
    m_pDIBOval = new CDIB;
    */

```

```

        if (!m_pDIBOval->Load(strFile))
        {
            delete m_pDIBOval;
            m_pDIBOval = NULL;
            return;
        }

        strFile = strPath + BMP_LOGIN_ANI;
        m_pDIBAni = new CPhasedSprite;
        if (!m_pDIBAni->Load(strFile))
        {
            delete m_pDIBAni;
            m_pDIBAni = NULL;
            return;        // skip
        }
        else
        {
#ifdef _MALL
            //
            m_pDIBAni->SetNumCells(1, 8); // Row, Col
            m_pDIBAni->SetNumCells(3, 4); // Row, Col
            m_pDIBAni->SetCell(0);
            m_pDIBAni->GetRect(m_rcAni);
            m_rcAni.OffsetRect(PTLT_ANI);
        }
    */
#endif

        strFile = strPath + BMP_LOGIN_BTN_OK;
        m_btnOK.Load(strFile);
        m_btnOK.SetPalette(m_pPal);

        strFile = strPath + BMP_LOGIN_BTN_NO;
        m_btnCancel.Load(strFile);
        m_btnCancel.SetPalette(m_pPal);

#ifdef _MALL
    /*
        for (int i=0; i < 2; i++)
        {
            strFile = (i==0) ? strPath + BMP_LOGIN_BTN_LAN
                           : strPath + BMP_LOGIN_BTN_MODEM;
            m_apDIBLM[i] = new CPhasedSprite;
            if (!m_apDIBLM[i]->Load(strFile))
            {
                delete m_apDIBLM[i];
                m_apDIBLM[i] = NULL;
            }
            else
            {
                m_apDIBLM[i]->SetNumCells(3, 1);    // Row, Col
                m_apDIBLM[i]->SetCell(0);
                m_apDIBLM[i]->GetRect(m_rcLM[i]);
                m_rcLM[i].OffsetRect(PTLT_LM[i]);
            }
        }
    */

    // m_nCurCell = 0;

```

LoginDlg.cpp

```
//      m_nBehavior      = ANI_TAP;
#endif
      m_uiTimer          = 0;
      m_nElapsedSec      = 0;
      m_dwLastTick       = 0L;

      m_NullBrush.CreateStockObject(NULL_BRUSH);
      m_fontMessage.CreateFont(-12, 0, 0, 0, FW_BOLD, // FW_NORMAL
                              FALSE, FALSE, 0, // bItalic, bUnderline, cStrikeOut
                              DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
                              DEFAULT_PITCH || FF_DONTCARE,
#ifdef _KOREAN
                              "±¼.²Ã¼");
#else
                              "Times New Roman");
#endif
}

CLoginDlg::~CLoginDlg()
{
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal)
        delete m_pPal;
#ifdef _MALL
/*    if (m_pDIBOval)
        delete m_pDIBOval;
    if (m_pDIBAni)
        delete m_pDIBAni;
    for (int i=0; i < 2; i++)
    {
        if (m_apDIBLM[i])
            delete m_apDIBLM[i];
    }*/
#endif
}

void CLoginDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CLoginDlg)
    DDX_CBString(pDX, IDC_CB_HOST, m_strHost);
    DDX_Text(pDX, IDC_EDIT_NICKNAME, m_strNickName);
    DDV_MaxChars(pDX, m_strNickName, 20);
    DDX_Text(pDX, IDC_EDIT_PASSWORD, m_strPassword);
    DDV_MaxChars(pDX, m_strPassword, 12);
    DDX_Text(pDX, IDC_ST_MESSAGE, m_strMessage);
    DDX_Text(pDX, IDC_ST_SECONDS, m_strSec);
    //}}AFX_DATA_MAP
}

void CLoginDlg::PositionControl(const int nCtrlID, const CRect& rcCtrl) const
{
    CWnd* pW = GetDlgItem(nCtrlID);
    ASSERT(pW);
    pW->SetWindowPos(NULL, rcCtrl.left, rcCtrl.top,
```



```

        rcCtrl.Width(), rcCtrl.Height(),
        SWP_NOZORDER | SWP_NOACTIVATE);
    }

    BOOL CLoginDlg::OnInitDialog()
    {
        if (!m_pDIBBack)
            return FALSE;
        CDialog::OnInitDialog();

        m_btnOK.SubclassDlgItem(IDOK, this);
        m_btnCancel.SubclassDlgItem(IDCANCEL, this);

#ifdef _MALL
        // CPoint ptLT(360, 262);
        // CPoint ptLT(338, 178);
        m_btnOK.MoveResize(ptLT);
        // ptLT.x = 10;
        // ptLT.y = 265;
        // ptLT.x = 266;
        // ptLT.y = 231;
        m_btnCancel.MoveResize(ptLT);
#else
        // CPoint ptLT(349, 238);
        // CPoint ptLT(35, 191);
        m_btnOK.MoveResize(ptLT);
        // ptLT.x = 17;
        // ptLT.x = 174;

        m_btnCancel.MoveResize(ptLT);
#endif

        PositionControl(IDC_EDIT_NICKNAME, RECT_NICKNAME);
        PositionControl(IDC_EDIT_PASSWORD, RECT_PASSWORD);
#ifdef _MALL
        // GetDlgItem(IDC_EDIT_PASSWORD)->ShowWindow(SW_HIDE);
#endif
        PositionControl(IDC_ST_MESSAGE, RECT_MESSAGE);
        PositionControl(IDC_ST_SECONDS, RECT_SECONDS);
        PositionControl(IDC_CB_HOST, RECT_HOST);

        CWnd* pW = GetDlgItem(IDC_ST_MESSAGE);
        pW->SetFont(&m_fontMessage, FALSE);

        CComboBox* pCB = (CComboBox*)GetDlgItem(IDC_CB_HOST);
        pCB->SetItemHeight(-1, 15);
        pCB->ShowWindow(SW_HIDE);
        for (int i=0; i < gResMan.GetNumServerIPs(); i++)
        {
            pCB->InsertString(-1, *gResMan.GetServerIP(i));
        }
        CUC2App* pApp = (CUC2App*)AfxGetApp();
        ASSERT(pApp);
        m_nTimeOut = pApp->RegGetTimeOut();
        if (m_nTimeOut < 10)
            m_nTimeOut = 10; // must be at least 10 sec.
    }

```

```

SetConnType(pApp->RegGetConnType());
m_uiTimer = SetTimer(4002, MSPT_TIMER, NULL);
// CG: The following block was added by the ToolTips component.
{
    // Create the ToolTip control.
    m_tooltip.Create(this);
    m_tooltip.Activate(TRUE);

    m_tooltip.AddTool(GetDlgItem(IDC_EDIT_NICKNAME),
IDC_EDIT_NICKNAME);
    m_tooltip.AddTool(GetDlgItem(IDC_EDIT_PASSWORD),
IDC_EDIT_PASSWORD);
//    m_tooltip.AddTool(this, IDS_CONNECT_LAN, &m_rcLM[0], 100);
//    m_tooltip.AddTool(this, IDS_CONNECT_MODEM, &m_rcLM[1], 101);
    // m_tooltip.AddTool(GetDlgItem(IDC_<name>), "<text>");
}
/*
CRect rcWnd;
GetWindowRect(&rcWnd);
CClientDC dc(this);
// Center this dialog in the screen
CPoint lt( (dc.GetDeviceCaps(HORZRES) - rcWnd.Width()) / 2,
            (dc.GetDeviceCaps(VERTRES) - rcWnd.Height()) / 2);
SetWindowPos(NULL, lt.x, lt.y, 0, 0, SWP_NOSIZE | SWP_NOZORDER |
SWP_NOACTIVATE);
*/
//    CSplashWnd::HideSplashScreen();
#ifdef _MALL
//    StartAnimation(ANI_TAP);
#endif
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CLoginDlg::OnPaletteChanged(CWnd* pFocusWnd)
{
    CDialog::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CLoginDlg::OnQueryNewPalette()
{
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE); //
foreground
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
//        if (u)
//        { // Some colors changed so we need to do a repaint.
//            Invalidate(); // Repaint the lot.

```

```

        return TRUE; // Say we did something.
    }
    return FALSE; // Say we did nothing.
}

BOOL CLoginDlg::PreTranslateMessage(MSG* pMsg)
{
    // CG: The following block was added by the ToolTips component.
    {
        // Let the ToolTip process this message.
        m_tooltip.RelayEvent(pMsg);
    }
    return CDialog::PreTranslateMessage(pMsg); // CG: This was added
by the ToolTips component.
}

void CLoginDlg::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);

    if (m_pDIBBack)
        SetWindowPos(NULL, 0, 0, m_pDIBBack->GetWidth(), m_pDIBBack-
>GetHeight(),
                        SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
}

BOOL CLoginDlg::OnEraseBkgnd(CDC* pDC)
{
    CDialog::OnEraseBkgnd(pDC); // This should be called here!

    // Make sure we have what we need to do a paint.
    if (!m_pDIBBack) //|| !m_pDIBOval)
    {
        TRACE("CLginDlg: No DIB!\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = pDC->SelectPalette(m_pPal, FALSE); //
bForceBackground = FALSE
        // dc.RealizePalette(); // we realize in response to
WM_QUERYNEWPALETTE
    }
    m_pDIBBack->Draw(pDC, 0, 0);
#ifdef _MALL
    // m_pDIBOval->Draw(pDC, PTLT_OVAL.x, PTLT_OVAL.y);
#endif
    /*
    if (m_pDIBAni)
        m_pDIBAni->Draw(pDC, PTLT_ANI);
    if (m_apDIBLM[0])
        m_apDIBLM[0]->Draw(pDC, PTLT_LM[0]);
    if (m_apDIBLM[1])

```

```

        m_apDIBLM[1]->Draw(pDC, PTLT_LM[1]);
*/
    // Select old palette if we altered it.
    if (pPalOld)
        pDC->SelectPalette(pPalOld, FALSE);

    return TRUE;
}

void CLoginDlg::OnPaint()    // will work for InvalidateRect()
{
    CPaintDC dc(this); // device context for painting

    // Make sure we have what we need to do a paint.
    if (!m_pDIBBack) // || !m_pDIBOval
    {
        TRACE("CLginDlg: No DIB!\n");
        return;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = dc.SelectPalette(m_pPal, FALSE);    //
bForceBackground = FALSE
        // dc.RealizePalette();    // we realize in response to
WM_QUERYNEWPALETTE
    }
    // m_pDIBBack->Draw(&dc, 0, 0);
    // m_pDIBOval->Draw(&dc, PTLT_OVAL.x, PTLT_OVAL.y);
#ifdef _MALL
    /* if (m_pDIBAni)
        m_pDIBAni->Draw(&dc, PTLT_ANI);
    if (m_apDIBLM[0])
        m_apDIBLM[0]->Draw(&dc, PTLT_LM[0]);
    if (m_apDIBLM[1])
        m_apDIBLM[1]->Draw(&dc, PTLT_LM[1]); */
#endif
    // Select old palette if we altered it.
    if (pPalOld)
        dc.SelectPalette(pPalOld, FALSE);
}

void CLoginDlg::OnTimer(UINT nIDEvent)
{
    if (m_bConnecting)
    {
        int nSec = (::GetTickCount() - m_dwLastTick)/1000;
        if (nSec != m_nElapsedSec)
        {
            m_nElapsedSec = nSec;
            CWnd* pW = GetDlgItem(IDC_ST_SECONDS);
            pW->ShowWindow(SW_HIDE);
            m_strSec.Format("%3d", m_nElapsedSec);
            UpdateData(FALSE);    // Write
            InvalidateRect(RECT_SECONDS, TRUE);
        }
    }
}

```

```

//          pW->ShowWindow(SW_SHOW);
          if (m_nElapsedSec > m_nTimeOut)
          {
              ShowMessage(IDS_ERROR_TIMEOUT);
              m_pSocket->FDDisconnect();
              StopCount();
              WaitForConnectThread(); // Wait until the thread ends
              m_btnOK.EnableWindow(TRUE);
              GetDlgItem(IDCANCEL);
              CWnd* pWnd = GetDlgItem(IDCANCEL);
              pWnd->SetFocus();
          }
      }

      //////////// Animation
#ifdef _MALL
/*      if (!m_pDIBAni || (m_nBehavior < 0) || m_bPause)
        return;

        if (++m_nCurCell > ANI_BEH_COUNT[m_nBehavior])
        {
            if (m_bConnecting)
                StartAnimation(ANI_ROUND);
            else
            {
                StartAnimation(ANI_TAP);
                StopAnimation();
            }
        }
*/
#endif

        CClientDC dc(this);
        CPalette* pPalOld = NULL;
        if (m_pPal)
        {
            pPalOld = dc.SelectPalette(m_pPal, FALSE); //
bForceBackground = FALSE
            // dc.RealizePalette(); // we realize in response to
WM_QUERYNEWPALETTE
        }

#ifdef _MALL
/*      int nCell = ANI_BEH[m_nBehavior][m_nCurCell];
        if (nCell != m_pDIBAni->GetCellID())
        {
            m_pDIBAni->SetCell(nCell);
            m_pDIBAni->Draw(&dc, PTLT_ANI);
        }
*/
#endif
        // Select old palette if we altered it.
        if (pPalOld)
            dc.SelectPalette(pPalOld, FALSE);

        //      CDialog::OnTimer(nIDEvent);

```

```

}

void CLoginDlg::OnDestroy()
{
    CDialog::OnDestroy();

    if (m_uiTimer)
    {
        KillTimer(m_uiTimer);
        m_uiTimer = 0;
    }
}

BOOL CLoginDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    // ::SetCursor(AfxGetApp()->LoadCursor(IDC_HARROW));
    // return TRUE;
    return CDialog::OnSetCursor(pWnd, nHitTest, message);
}

void CLoginDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
#ifdef _MALL
    /* m_nBehavior++;
       if (m_nBehavior > 2)
           m_nBehavior = 0;
       m_nCurCell = 0;
       if (!m_bConnecting && m_apDIBLM[0] && m_apDIBLM[1])
       {
           if (m_rcLM[0].PtInRect(point)) // LAN
           {
               SetConnType(GLOBAL_HOST);
               ShowMessage(IDS_CONNECTION_TYPE_GLOBAL);
           }
           else if (m_rcLM[1].PtInRect(point)) // MODEM
           {
               SetConnType(LOCAL_HOST);
               ShowMessage(IDS_CONNECTION_TYPE_LOCAL);
           }
       }
    */
#endif
    CDialog::OnLButtonDown(nFlags, point);
}

void CLoginDlg::SetConnType(const int n)
{
    m_nConnType = n;
    if (m_nConnType == USERDEF_HOST)
    {
        CUC2App* pApp = (CUC2App*)AfxGetApp();
        ASSERT(pApp);
        pApp->RegGetServer(m_strHost);
        CComboBox* pCB = (CComboBox*)GetDlgItem(IDC_CB_HOST);
        pCB->ShowWindow(SW_SHOW);
        if (!m_strHost.IsEmpty())
        {

```

```

        pCB->InsertString(0, m_strHost);
        pCB->SetCurSel(0);
    }
}

#ifdef _MALL
/*      if (m_apDIBLM[0] && m_apDIBLM[1])
    {
        m_apDIBLM[0]->SetCell((n == GLOBAL_HOST) ? 2 : 0);
        m_apDIBLM[1]->SetCell((n == LOCAL_HOST) ? 2 : 0);
        InvalidateRect(m_rcLM[0], FALSE);
        InvalidateRect(m_rcLM[1], FALSE);
    }
*/
#endif

void CLoginDlg::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    if (nFlags & MK_CONTROL)
    {
#ifdef _MALL
        // Authority Check
        if (m_rcAni.PtInRect(point))
        {
            CInputPassword PasswordDlg;
            PasswordDlg.DoModal();
            if (PasswordDlg.m_strPassword == UC2PASSWORD)
            {
                CComboBox* pCB = (CComboBox*)GetDlgItem(IDC_CB_HOST);
                pCB->ShowWindow(SW_SHOW);
                SetConnType(USERDEF_HOST);
            }
        }
#endif
    }
    CDialog::OnLButtonDblClk(nFlags, point);
}

void CLoginDlg::OnOK()
{
    m_btnOK.EnableWindow(FALSE); // Disable OK button to prohibit
    reentrancy
    UpdateControlBackground(RECT_MESSAGE);
    UpdateData(TRUE); // Retrieve
    if (m_strNickName.IsEmpty())
    {
        //      MsgProgress2("Ã¸Ã¸ ÀÌ.ŞÀ» ÀÔ.ÂÇI¸¸a.");
        CEdit* pEdit = (CEdit*)GetDlgItem(IDC_EDIT_NICKNAME);
        pEdit->SetFocus();
        pEdit->SetSel(0, -1); // select all
        pEdit->ReplaceSel("chat-name!");
        pEdit->SetSel(0, -1);
        m_btnOK.EnableWindow(TRUE);
        return;
    }
    ShowMessage(IDS_LOGIN_CONNECTING);
#ifdef _MALL

```

```

//      StartAnimation(ANI_ROUND);
#endif

// New Connect
::ZeroMemory(&m_ecInfo, sizeof(EC_CONNINFO));
int len;

CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
CUC2Doc* pDoc = (CUC2Doc*)pMF->GetActiveDocument();

pDoc->PlayWaveFile("down_event.wav");

CUC2App* pApp = (CUC2App*)AfxGetApp();
ASSERT(pApp);
pApp->RegSetConnType(m_nConnType); // Save current connection type

if (m_nConnType == USERDEF_HOST)
{
    len = m_strHost.GetLength();
    m_ecInfo.szServer = m_strHost.GetBuffer(len);
    pApp->RegSetServer(m_strHost); // Save server address
}
else
{
    CString* pS = gResMan.GetServerIP(m_nConnType);
    // IP address
    *pS = "216.112.5.150";

    if (pS)
    {
        len = pS->GetLength();
        m_ecInfo.szServer = pS->GetBuffer(len);
    }
    else
    {
        m_ecInfo.szServer = _T("88.1.26.2");
    }
}

len = m_strNickName.GetLength();
m_ecInfo.szNick = m_strNickName.GetBuffer(len);
// This should be in memory for ConnectionFunction thread
static CString strNickBak = m_strNickName + _T("0");
m_ecInfo.szNickBak = strNickBak.GetBuffer(len+1);
static CString strUserName;
strUserName.Format("%c%d%02d", UC2_SIGN_CHAR, GetClientVersion() /
100, GetClientVersion() % 100);
// strUserName += *((CMainFrame*)AfxGetMainWnd())->GetUserID(); //
Can't use Korean
len = strUserName.GetLength();
m_ecInfo.szUserName = strUserName.GetBuffer(len+1);
m_ecInfo.szPass = NULL;
m_ecInfo.fAuthenticate = FALSE; // use anonymous connection
ASSERT(m_nTimeout >= 10);
m_ecInfo.dwTimeout = (DWORD)m_nTimeout * 1000; //
milliseconds

```



```

HRESULT hr = ::HrVerifyNickA(m_ecInfo.szNick);
//HRESULT hr = ::HrVerifyNickW(m_ecInfo.szNick);
if (FAILED(hr))
{
    ShowMessage(IDS_INVALID_CHATID);
    CEdit* pEdit = (CEdit*)GetDlgItem(IDC_EDIT_NICKNAME);
    pEdit->SetFocus();
    pEdit->SetSel(0, -1); // select all
    m_btnOK.EnableWindow(TRUE);
    return;
}

StartCount();
// m_btnCancel.EnableWindow(FALSE);
WaitForConnectThread();
m_pConnThread = AfxBeginThread(ConnectFunction, (LPVOID)this);
ASSERT(m_pConnThread);
m_hConnThread = m_pConnThread->m_hThread; // Save the handle for
::WaitSingleObject
TRACE("CLoginDlg created a thread for ConnectionFunction [%lx]\n",
      m_pConnThread->m_nThreadID);

// CDialog::OnOK(); // ConnectionFunction will call EndDialog
}

void CLoginDlg::OnCancel()
{
    ShowMessage(IDS_CANCELED_CONNECTION);

    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    CUC2Doc* pDoc = (CUC2Doc*)pMF->GetActiveDocument();

    pDoc->PlayWaveFile("down_event.wav");

    m_pSocket->FDDisconnect();
    StopCount();
    WaitForConnectThread(); // Wait until the thread ends

    CDialog::OnCancel();
}

HBRUSH CLoginDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
#ifdef _MALL
    if (nCtlColor != CTLCOLOR_EDIT)
    {
        pDC->SetTextColor(PALETTERGB(0, 0, 255)); // yellow 255,255,128;
        pDC->SetBkMode(TRANSPARENT);
        return (HBRUSH)m_NullBrush;
    }
    else
    {
        return CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
    }
}
#else
    pDC->SetBkMode(TRANSPARENT);
#endif
}

```

```

        return (HBRUSH)m_NullBrush;
#endif
}

void CLoginDlg::UpdateControlBackground(const CRect& rcBack, const bDeflate)
{
    if (!m_pDIBBack)
        return;
    CRect rc(rcBack);
    if (bDeflate)
        rc.DeflateRect(1, 2);
    InvalidateRect(&rc, TRUE);
}

void CLoginDlg::ShowMessage(const int nIDS)
{
    UpdateData(TRUE); // Retrieve contents in edit control
    if (nIDS && !m_strMessage.LoadString(nIDS))
        return;
    CWnd* pWnd = GetDlgItem(IDC_ST_MESSAGE);
    pWnd->ShowWindow(SW_HIDE);
    UpdateControlBackground(RECT_MESSAGE, FALSE); // Don't deflate
rectangle
    UpdateData(FALSE); // Write
    pWnd->ShowWindow(SW_SHOW);
}

void CLoginDlg::OnSetfocusEditNickname()
{
    if (m_bFirstFocus)
    {
        m_bFirstFocus = FALSE;
        return;
    }
    ShowMessage(IDC_EDIT_NICKNAME);
#ifdef _MALL
    // StartAnimation(ANI_LOOK);
#endif
}

void CLoginDlg::OnSetfocusEditPassword()
{
    ShowMessage(IDC_EDIT_PASSWORD);
#ifdef _MALL
    // StartAnimation(ANI_LOOK);
#endif
}

void CLoginDlg::OnUpdateEditNickname()
{
    // TODO: If this is a RICHEDIT control, the control will not
    // send this notification unless you override the
CDialog::OnInitDialog()
    // function to send the EM_SETEVENTMASK message to the control
    // with the ENM_UPDATE flag Ored into the lParam mask.
#ifdef _MALL
    // StartAnimation(ANI_LOOK);

```

LoginDlg.cpp

```
#endif
    UpdateControlBackground(RECT_NICKNAME);
}

void CLoginDlg::OnKillfocusEditNickname()
{
    OnUpdateEditNickname();
}

void CLoginDlg::OnUpdateEditPassword()
{
    UpdateControlBackground(RECT_PASSWORD);
}

void CLoginDlg::OnKillfocusEditPassword()
{
    OnUpdateEditPassword();
}

void CLoginDlg::OnEditupdateCbHost()
{
    CRect rc(RECT_HOST);
    rc.bottom = rc.top + 22;
    UpdateControlBackground(rc);
}

void CLoginDlg::OnKillfocusCbHost()
{
    OnEditupdateCbHost();
}

// another thread for connection
// m_pFactory->HrMakeSocket() blocks
UINT CLoginDlg::ConnectFunction(LPVOID pParam)
{
    CLoginDlg* pDlg = (CLoginDlg*)pParam;
    HWND hDlg = pDlg->GetSafeHwnd();
    if (!pDlg->m_pSocket->FConnect(&pDlg->m_ecInfo, hDlg))
    {
        ::PostMessage(hDlg, WM_COMMAND, MAKEWPARAM(CMD_CONNECT_FAILURE,
0), 0);
        return FALSE;
    }
    pDlg->EndDialog(IDOK);

    return TRUE;
}

void CLoginDlg::WaitForConnectThread() // to end
{
    if (!m_hConnThread)
        return;

    TRACE("CLoginDlg - ::WaitForSingleObject(0x%lx, 3000L);\n",
m_hConnThread);
}
```

```

        DWORD dwRc = ::WaitForSingleObject(m_hConnThread, 3000L);
        switch (dwRc)
        {
        case WAIT_OBJECT_0:
            TRACE0("Thread: WAIT_OBJECT_0\n");
            break;
        case WAIT_ABANDONED:
            TRACE0("Thread: WAIT_ABANDONED\n");
            break;
        case WAIT_TIMEOUT:
            TRACE0("Thread Hung! Deleting...\n");
            AfxMessageBox("Connecting thread hung!");
            delete m_pConnThread;
            break;
        case WAIT_FAILED:
            PrintWin32Error("Thread: WAIT_FAILED ");
            break;
        default:
            TRACE0("Thread Hung!\n");
            break;
        }
        m_pConnThread = NULL;    // auto-deleted
        m_hConnThread = NULL;
    }

void CLoginDlg::StartCount()
{
    m_dwLastTick      = ::GetTickCount();
    m_nElapsedSec     = -1;
    m_bConnecting     = TRUE;
}

void CLoginDlg::StopCount()
{
    m_bConnecting     = FALSE;
}

////////////////////////////////////
/////
// ChatSock message handler

LRESULT CLoginDlg::OnConnectConnecting(WPARAM wParam, LPARAM lParam)
{
    ShowMessage(IDS_CONNECTING_SERVER);
    return 0;
}

LRESULT CLoginDlg::OnConnectLogin(WPARAM wParam, LPARAM lParam)
{
    ShowMessage(IDS_LOGGING_ON_SERVER);
    return 0;
}

LRESULT CLoginDlg::OnConnectBackupID(WPARAM wParam, LPARAM lParam)
{
    ShowMessage(IDS_TRYING_BACKUP_ID);
    m_ecInfo.szNick = m_ecInfo.szNickBak;
}

```

```

        return 0;
    }
LRESULT CLoginDlg::OnConnectFailure(WPARAM wParam, LPARAM lParam)
{
    ShowMessage(IDS_CONNECTION_FAILED);
    // AfxGetApp()->WinHelp(HID_MYTOPIC);
    StopCount();
    m_btnOK.EnableWindow(TRUE);
    // m_bmbCancel.EnableWindow(TRUE);
    return 0;
}

UINT CLoginDlg::OnNcHitTest(CPoint point)
{
    UINT nHitTest = CDialog::OnNcHitTest(point);
#ifdef _MALL
    CPoint pt(point);
    ScreenToClient(&pt);
    // const CRect rcInput(129, 172, 366, 261);
    const CRect rcInput(113, 159, 307, 246);
    if (!rcInput.PtInRect(pt) && !m_rcAni.PtInRect(pt) &&
        (nHitTest == HTCLIENT) && (::GetAsyncKeyState(MK_LBUTTON) < 0))
        nHitTest = HTCAPTION;
#endif
    return nHitTest;
}

```

LoginDlg.h

```

#if !defined(AFX_LOGINDLG_H__5D07DF04_B344_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_LOGINDLG_H__5D07DF04_B344_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// LoginDlg.h : header file
//
#include "UC2Ani/PSButton.h"
#include "UC2CS.h" // EC_CONNINFO
////////////////////////////////////
// CLoginDlg dialog
class CDIB;
class CDIBPal;
class CPhasedSprite;
class CUC2Socket;

enum CONNECTION_TYPE
{
    GLOBAL_HOST=0, LOCAL_HOST=1, USERDEF_HOST
};

class CLoginDlg : public CDialog
{
// Construction
public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    CLoginDlg(CUC2Socket* pSocket, CWnd* pParent=NULL); // standard
    constructor
    virtual ~CLginDlg();

#ifdef _MALL
    enum ANI_BEHAVIOR
    {
        ANI_TAP=0, ANI_LOOK, ANI_ROUND
    };
#endif

// Dialog Data
    //{AFX_DATA(CLoginDlg)
    enum { IDD = IDD_DIALOG_LOGIN };
    CString      m_strHost;
    CString      m_strNickName;
    CString      m_strPassword;
    CString      m_strMessage;
    CString      m_strSec;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CLoginDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

```

LoginDlg.h

```

        void                PositionControl(const int nCtrlID, const CRect& rcCtrl)
const;
        void                UpdateControlBackground(const CRect& rcBack, const
bDeflate=TRUE);
        void                ShowMessage(const int nIDS);
        void                SetConnType(const int n);

        CToolTipCtrl        m_tooltip;
        CDIB*               m_pDIBBack;           // Background frame image
#ifdef _MALL
        CDIB*               m_pDIBOval;           // Oval image
        CPhasedSprite*      m_pDIBAni;           // Ani Character
        CPhasedSprite*      m_apDIBLM[2];         // LAN or MODEM button image
        CRect               m_rcAni;
        CRect               m_rcLM[2];           // LAN or MODEM
#endif

        CDIBPal*            m_pPal;               // main palette
        CPSButton            m_btnOK;
        CPSButton            m_btnCancel;
        CBrush               m_NullBrush;
        BOOL                m_bFirstFocus;

#ifdef _MALL
// Animation
        void                StopAnimation()
                        { m_bPause = TRUE; m_nCurCell = 0; }

        void                StartAnimation(const int nBeh)
                        { m_bPause = FALSE; m_nBehavior = nBeh; m_nCurCell =
0; }

        int                m_nCurCell;           // for m_pDIBAni
        int                m_nBehavior;           // for m_pDIBAni
        BOOL               m_bPause;             // Pause Animation
#endif

        UINT               m_uiTimer;
        int                m_nElapsedSec;
        DWORD              m_dwLastTick;
        CFont              m_fontMessage;

        CUC2Socket*         m_pSocket;
        EC_CONNINFO         m_ecInfo;
        BOOL               m_bConnecting;
        CWinThread*         m_pConnThread;
        HANDLE              m_hConnThread;
        int                m_nConnType;
        int                m_nTimeOut;

        void                StartCount();
        void                StopCount();
        static UINT ConnectFunction(LPVOID pParam);
        void                WaitForConnectThread();

// ON_MESSAGE
LRESULT OnConnectConnecting(WPARAM, LPARAM);
LRESULT OnConnectLogin(WPARAM, LPARAM);
LRESULT OnConnectBackupID(WPARAM, LPARAM);

```

```

LRESULT OnConnectFailure(WPARAM, LPARAM);

// Generated message map functions
//{{AFX_MSG(CLoginDlg)
virtual BOOL OnInitDialog();
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
afx_msg BOOL OnQueryNewPalette();
afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
afx_msg void OnSize(UINT nType, int cx, int cy);
afx_msg void OnTimer(UINT nIDEvent);
afx_msg void OnDestroy();
afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
afx_msg void OnSetfocusEditNickname();
afx_msg void OnSetfocusEditPassword();
afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
virtual void OnOK();
virtual void OnCancel();
afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
afx_msg void OnUpdateEditNickname();
afx_msg void OnUpdateEditPassword();
afx_msg void OnKillfocusEditNickname();
afx_msg void OnKillfocusEditPassword();
afx_msg void OnEditupdateCbHost();
afx_msg void OnKillfocusCbHost();
afx_msg void OnPaint();
afx_msg UINT OnNcHitTest(CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_LOGINDLG_H__5D07DF04_B344_11D1_80E2_080009B9F339__INCLUDED_)

```


| Year | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 | 2096 | 2097 | 2098 | 2099 | 2100 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|
| Population | 1,200,000 | 1,250,000 | 1,300,000 | 1,350,000 | 1,400,000 | 1,450,000 | 1,500,000 | 1,550,000 | 1,600,000 | 1,650,000 | 1,700,000 | 1,750,000 | 1,800,000 | 1,850,000 | 1,900,000 | 1,950,000 | 2,000,000 | 2,050,000 | 2,100,000 | 2,150,000 | 2,200,000 | 2,250,000 | 2,300,000 | 2,350,000 | 2,400,000 | 2,450,000 | 2,500,000 | 2,550,000 | 2,600,000 | 2,650,000 | 2,700,000 | 2,750,000 | 2,800,000 | 2,850,000 | 2,900,000 | 2,950,000 | 3,000,000 | 3,050,000 | 3,100,000 | 3,150,000 | 3,200,000 | 3,250,000 | 3,300,000 | 3,350,000 | 3,400,000 | 3,450,000 | 3,500,000 | 3,550,000 | 3,600,000 | 3,650,000 | 3,700,000 | 3,750,000 | 3,800,000 | 3,850,000 | 3,900,000 | 3,950,000 | 4,000,000 | 4,050,000 | 4,100,000 | 4,150,000 | 4,200,000 | 4,250,000 | 4,300,000 | 4,350,000 | 4,400,000 | 4,450,000 | 4,500,000 | 4,550,000 | 4,600,000 | 4,650,000 | 4,700,000 | 4,750,000 | 4,800,000 | 4,850,000 | 4,900,000 | 4,950,000 | 5,000,000 | 5,050,000 | 5,100,000 | 5,150,000 | 5,200,000 | 5,250,000 | 5,300,000 | 5,350,000 | 5,400,000 | 5,450,000 | 5,500,000 | 5,550,000 | 5,600,000 | 5,650,000 | 5,700,000 | 5,750,000 | 5,800,000 | 5,850,000 | 5,900,000 | 5,950,000 | 6,000,000 | 6,050,000 | 6,100,000 | 6,150,000 | 6,200,000 | 6,250,000 | 6,300,000 | 6,350,000 | 6,400,000 | 6,450,000 | 6,500,000 | 6,550,000 | 6,600,000 | 6,650,000 | 6,700,000 | 6,750,000 | 6,800,000 | 6,850,000 | 6,900,000 | 6,950,000 | 7,000,000 | 7,050,000 | 7,100,000 | 7,150,000 | 7,200,000 | 7,250,000 | 7,300,000 | 7,350,000 | 7,400,000 | 7,450,000 | 7,500,000 | 7,550,000 | 7,600,000 | 7,650,000 | 7,700,000 | 7,750,000 | 7,800,000 | 7,850,000 | 7,900,000 | 7,950,000 | 8,000,000 | 8,050,000 | 8,100,000 | 8,150,000 | 8,200,000 | 8,250,000 | 8,300,000 | 8,350,000 | 8,400,000 | 8,450,000 | 8,500,000 | 8,550,000 | 8,600,000 | 8,650,000 | 8,700,000 | 8,750,000 | 8,800,000 | 8,850,000 | 8,900,000 | 8,950,000 | 9,000,000 | 9,050,000 | 9,100,000 | 9,150,000 | 9,200,000 | 9,250,000 | 9,300,000 | 9,350,000 | 9,400,000 | 9,450,000 | 9,500,000 | 9,550,000 | 9,600,000 | 9,650,000 | 9,700,000 | 9,750,000 | 9,800,000 | 9,850,000 | 9,900,000 | 9,950,000 | 10,000,000 |

A-186



```

ON_COMMAND(ID_EDIT_CUT, OnEditCut)
ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, OnUpdateEditCut)
ON_COMMAND(ID_EDIT_NOTEPAD, OnEditNotepad)
ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
ON_COMMAND(ID_EDIT_UNDO, OnEditUndo)
ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, OnUpdateEditUndo)
ON_COMMAND(ID_HELP_HOMEPAGE, OnHelpHomepage)
// SATYA CHANGES START
// Added new message Handlers for PopMenu from
// desk top Icon..
ON_COMMAND(UCMN_MINIMISE, OnPopMinimise)
ON_COMMAND(UCMN_MAXIMISE, OnPopMaximise)
ON_COMMAND(UCMN_UNICHAT, OnUniChat)
ON_COMMAND(UCMN_EXIT, OnUCExit)
ON_WM_SYSCOMMAND()
// SATYA CHANGES END
//}}AFX_MSG_MAP
// Global help commands
ON_COMMAND(ID_HELP_FINDER, CFrameWnd::OnHelpFinder)
ON_COMMAND(ID_HELP, CFrameWnd::OnHelp)
ON_COMMAND(ID_CONTEXT_HELP, CFrameWnd::OnContextHelp)
ON_COMMAND(ID_DEFAULT_HELP, CFrameWnd::OnHelpFinder)
// User-Defined Messages
ON_MESSAGE(CSMMSG_CMD_ADDCHANNEL, OnCsAddChannel)
ON_MESSAGE(CSMMSG_CMD_PRIVATEMSG, OnCsPrivateMsg)
ON_MESSAGE(CSMMSG_CMD_QUERYDATA, OnCsQueryData)
ON_MESSAGE(CSMMSG_CMD_INVITE, OnCsInvite)
ON_MESSAGE(CSMMSG_CMD_GOTMEMLIST, OnCsGotMemList)
ON_MESSAGE(CSMMSG_CMD_ADDMEMBER, OnCsAddMember)
ON_MESSAGE(CSMMSG_CMD_DELMEMBER, OnCsDelMember)
ON_MESSAGE(CSMMSG_CMD_DELCHANNEL, OnCsDelChannel)
ON_MESSAGE(CSMMSG_CMD_MODEMEMBER, OnCsModeMember)
ON_MESSAGE(CSMMSG_CMD_MODECHANNEL, OnCsModeChannel)
ON_MESSAGE(CSMMSG_CMD_TEXT_A, OnCsTextA)
ON_MESSAGE(CSMMSG_CMD_DATA, OnCsData)
ON_MESSAGE(CSMMSG_CMD_WHISPERTEXT_A, OnCsWhisperText)
ON_MESSAGE(CSMMSG_CMD_WHISPERDATA, OnCsWhisperData)
ON_MESSAGE(CSMMSG_CMD_NEWTOPIC, OnCsNewTopic)
ON_MESSAGE(CSMMSG_CMD_NEWNICK, OnCsNewNick)
ON_MESSAGE(CMD_CHANNELFULL_RETRY, OnChannelFullRetry)
#ifdef _UNITEL
ON_MESSAGE(WM_COMMREAD, OnCommRead)
ON_MESSAGE(UM_EXIT, OnUmExit)
// ON_MESSAGE(UM_DOWNLOAD, OnUmDownload)
#endif // _UNITEL
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR, // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

////////////////////////////////////

```

```
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    TRACE("CMainFrame::CMainFrame()\n");
    m_tmStart = CTime::GetCurrentTime();
    m_pPalette = NULL;
    m_bFreeze = FALSE;
    m_bAskBeforeClose = TRUE;
}

CMainFrame::~CMainFrame()
{
    TRACE("CMainFrame::~CMainFrame()\n");
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    TRACE("CMainFrame::OnCreate()\n");
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    /*
    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;        // fail to create
    }
    */

    /*
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;        // fail to create
    }
    */

    // TODO: Remove this if you don't want tool tips or a resizable
    toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    // be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    /*
    // TODO: Add a menu item that will toggle the visibility of the
    // dialog bar named "ControlPanel":
    // 1. In ResourceView, open the menu resource that is used by
    // the CMainFrame class
    // 2. Select the View submenu
    // 3. Double-click on the blank item at the bottom of the submenu
    // 4. Assign the new item an ID: CG_ID_VIEW_CONTROLPANEL
    // 5. Assign the item a Caption: ControlPanel
    */

```

```

// TODO: Change the value of CG_ID_VIEW_CONTROLPANEL to an appropriate
value:
// 1. Open the file resource.h
// CG: The following block was inserted by the 'Dialog Bar' component
{
    // Initialize dialog bar m_wndControlPanel
    if (!m_wndControlPanel.Create(this,
        CBRS_BOTTOM | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_HIDE_INPLACE))
    {
        TRACE0("Failed to create dialog bar m_wndControlPanel\n");
        return -1;        // fail to create
    }
#ifdef DOCKING
    m_wndControlPanel.EnableDocking(CBRS_ALIGN_BOTTOM |
CBRS_ALIGN_TOP);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndControlPanel);
#endif
}

// TODO: Add a menu item that will toggle the visibility of the
// dialog bar named "HistoryPanel":
// 1. In ResourceView, open the menu resource that is used by
// the CMainFrame class
// 2. Select the View submenu
// 3. Double-click on the blank item at the bottom of the submenu
// 4. Assign the new item an ID: CG_ID_VIEW_HISTORYPANEL
// 5. Assign the item a Caption: HistoryPanel

// TODO: Change the value of CG_ID_VIEW_HISTORYPANEL to an appropriate
value:
// 1. Open the file resource.h
// CG: The following block was inserted by the 'Dialog Bar' component
{
    // Initialize dialog bar m_wndHistoryPanel
    if (!m_wndHistoryPanel.Create(this,
        CBRS_RIGHT | CBRS_TOOLTIPS | CBRS_FLYBY |
CBRS_HIDE_INPLACE))
    {
        TRACE0("Failed to create dialog bar m_wndHistoryPanel\n");
        return -1;        // fail to create
    }

    m_wndHistoryPanel.EnableDocking(CBRS_ALIGN_RIGHT |
CBRS_ALIGN_LEFT);
    EnableDocking(CBRS_ALIGN_ANY);
    // Changes Start for Out
    EnableDocking(CBRS_ALIGN_ANY);
    //DockControlBar(&m_wndHistoryPanel); // syc 0708

    // syc 0708 begin
    CRect rect;
    GetWindowRect(&rect);
    CPoint pt(rect.right - 40, rect.top + 25);
    FloatControlBar(&m_wndHistoryPanel, pt);
    // end

```

```

        //DockControlBar(&m_wndHistoryPanel);
        // Changes End !!!!
        CString strTitle;
        if (strTitle.LoadString(IDS_HISTORY_PANEL))
            m_wndHistoryPanel.SetWindowText(strTitle);

        ShowControlBar(&m_wndHistoryPanel, FALSE, FALSE);
    }

    // CG: The following line was added by the Splash Screen component.
    TRACE("CSplashWnd::ShowSplashScreen()\n");
    CSplashWnd::ShowSplashScreen(this);
    TRACE("CMainFrame::OnCreate() - done\n");

    // CG: The following block was inserted by 'Status Bar' component.
    {
        // Find out the size of the static variable 'indicators' defined
        // by AppWizard and copy it
        int nOrigSize = sizeof(indicators) / sizeof(UINT);

        UINT* pIndicators = new UINT[nOrigSize + 2];
        memcpy(pIndicators, indicators, sizeof(indicators));

        // Call the Status Bar Component's status bar creation function
        if (!InitStatusBar(pIndicators, nOrigSize, 60))
        {
            TRACE0("Failed to initialize Status Bar\n");
            return -1;
        }
        delete[] pIndicators;
    }

    //SATYA CHANGES START
    // add Unichat icon on the status area
    if (!AddUCIconOnStatusArea())
    {
        AfxMessageBox("Add ICon Failed");
    }
    //SATYA CHANGES END

    return 0;
}

void CMainFrame::ShowHistoryPanel(const BOOL bShow)
{
    ShowControlBar(&m_wndHistoryPanel, bShow, FALSE);
}

void CMainFrame::AdjustFrame(const int nW, const int nH)
{
    SetFreeze(FALSE);
    CClientDC dc(this);
    int nWidth = min(dc.GetDeviceCaps(HORZRES) - 40, nW);
    int nHeight = min(dc.GetDeviceCaps(VERTRES) - 40, nH);
    if (((nW + 40) >= dc.GetDeviceCaps(HORZRES)) ||

```

```

        ((nH + 40) >= dc.GetDeviceCaps(VERTRES)))
        SetWindowPos(&wndTopMost, 0, 0, nWidth, nHeight, SWP_NOZORDER);
    else
        SetWindowPos(&wndTopMost, 0, 0, nWidth, nHeight, SWP_NOMOVE |
SWP_NOZORDER);
    RecalcLayout();
    CScrollView* pView = (CScrollView*)GetActiveView();
    pView->ResizeParentToFit(FALSE);    // Try shrinking first
    pView->ResizeParentToFit(TRUE);     // Let's be daring
    SetFreeze();
}

// You can call this method in CXXView::OnInitialUpdate
BOOL CMainFrame::InitControlPanel(CPalette* pPal)
{
    TRACE0("CMainFrame::InitControlPanel()\n");
    SetPalette(pPal);
    if (!m_wndControlPanel.InitControls())
        return FALSE;
    if (!m_wndHistoryPanel.InitControls())
        return FALSE;
    return TRUE;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    TRACE0("CMainFrame::PreCreateWindow()\n");
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
    cs.cx = 640;
    cs.cy = 480;

#ifdef DEMO
    if (cs.hMenu)
    {
        ::DestroyMenu(cs.hMenu);
        cs.hMenu = NULL;
    }
#endif
    return CFrameWnd::PreCreateWindow(cs);
}

////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif // _DEBUG

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

BOOL CMainFrame::OnQueryNewPalette()
{
    CView* pview = GetActiveView();
    if (pview)
        pview->SendMessage(WM_QUERYNEWPALETTE, (WPARAM)0, (LPARAM)0);

    if (m_wndControlPanel.GetSafeHwnd())
        m_wndControlPanel.SendMessage(WM_QUERYNEWPALETTE, (WPARAM)0,
(LPARAM)0);

    return CFrameWnd::OnQueryNewPalette();
}

void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
    // CG: This function was added by the Palette Support component
    if (pFocusWnd == this || IsChild(pFocusWnd))
        return;

    // OnQueryNewPalette();

    // Pass this message on to the active view (OSBVeiw-derived)
    CView* pview = GetActiveView();
    if (pview) // OnPaletteChanged is not public, so I'll send a message
        pview->SendMessage(WM_PALETTECHANGED,
(WPARAM) (pFocusWnd->GetSafeHwnd()),
(LPARAM)0);

    if (m_wndControlPanel.GetSafeHwnd())
        m_wndControlPanel.SendMessage(WM_PALETTECHANGED,
(WPARAM) (pFocusWnd->GetSafeHwnd()), (LPARAM)0);
    CFrameWnd::OnPaletteChanged(pFocusWnd);
}

CPalette* CMainFrame::SetPalette(CPalette* pPalette)
{
    // CG: This function was added by the Palette Support component

    // Call this function when the palette changes. It will
    // realize the palette in the foreground to cause the screen
    // to repaint correctly. All calls to CDC::SelectPalette in
    // painting code should select palettes in the background.

    CPalette* pOldPalette = m_pPalette;
    m_pPalette = pPalette;
    OnQueryNewPalette();
    return pOldPalette;
}

void CMainFrame::OnClose()
{
    TRACE0("CMainFrame::OnClose()\n");
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    ASSERT(pDoc);
}

```

```

    if (pDoc->IsDemo())
    {
        pDoc->EndDemo();
        return;
    }
    if (m_bAskBeforeClose)
    {
        CCloseDlg dlg;
        dlg.m_strMessage.LoadString(IDS_CLOSE_TEXT);
        if (dlg.DoModal() != IDOK)
            return;
    }
    // delet the Icon before closing the application
    DeleteUCIconOnStatusArea();

    CFrameWnd::OnClose();
}

void CMainFrame::OnDestroy()
{
    TRACE0("CMainFrame::OnDestroy()\n");
    CFrameWnd::OnDestroy();
}

void CMainFrame::SetFreeze(const BOOL bFreeze)
{
    m_bFreeze = bFreeze;
    if (m_bFreeze)
        GetWindowRect(&m_rcMax);
}

void CMainFrame::OnGetMinMaxInfo(MINMAXINFO FAR* lpMMI)
{
    TRACE0("OnGetMinMaxInfo ");
    if (m_bFreeze)
    {
        CRect rc;
        GetWindowRect(&rc);
        TRACE("(%d,%d) (%d,%d)\n", rc.left, rc.top, m_rcMax.Width(),
m_rcMax.Height());
        lpMMI->ptMaxSize.x = m_rcMax.Width();
        lpMMI->ptMaxSize.y = m_rcMax.Height();
        lpMMI->ptMaxPosition.x = rc.left;
        lpMMI->ptMaxPosition.y = rc.top;
        lpMMI->ptMinTrackSize.x = 0;
        lpMMI->ptMinTrackSize.y = 0;
        lpMMI->ptMaxTrackSize.x = m_rcMax.Width();
        lpMMI->ptMaxTrackSize.y = m_rcMax.Height();
    }
    TRACE0("\n");
    CFrameWnd::OnGetMinMaxInfo(lpMMI);
}

// If the mainframe is maximized, disable the Freeze menu option because
// there is no point in making the window non-resizable because a maximized
// window cannot be resized anyway.
void CMainFrame::OnSize(UINT nType, int cx, int cy)

```



```

{
    CFrameWnd::OnSize(nType, cx, cy);
#ifdef _DEBUG
    TCHAR* szType = "";
    switch(nType)
    {
        case SIZE_MAXIMIZED:    szType = "SIZE_MAXIMIZED";    break;
        case SIZE_MINIMIZED:    szType = "SIZE_MINIMIZED";    break;
        case SIZE_RESTORED:     szType = "SIZE_RESTORED";     break;
        case SIZE_MAXHIDE:      szType = "SIZE_MAXHIDE";      break;
        case SIZE_MAXSHOW:      szType = "SIZE_MAXSHOW";      break;
    }
    TRACE("CMainFrame::OnSize(%s,%d,%d)\n", szType, cx, cy);
#endif

//    CMenu* pMenu = GetMenu();
//    if (nType == SIZE_MAXIMIZED)
//        pMenu->EnableMenuItem(ID_VIEW_FREEZE, MF_DISABLED | MF_GRAYED);
//    else
//        pMenu->EnableMenuItem(ID_VIEW_FREEZE, MF_ENABLED);
//    DrawMenuBar();
}

////////////////////////////////////
// Edit

HWND CMainFrame::GetNotepad()
{
    HWND hwndFind = ::FindWindow("Notepad", NULL);    // "Tortoise's
Experiment"
    CTime time = CTime::GetCurrentTime();
    CString strFile(time.Format(_T("UC%y%m%d.txt")));
    if (!hwndFind)
    {
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        ::ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        int len = strFile.GetLength();
        char* szFile = strFile.GetBuffer(len);
        HINSTANCE hRes = ::ShellExecute(GetSafeHwnd(), "open", szFile,
NULL, NULL, SW_SHOWNORMAL);
        TRACE("::ShellExecute(%xh) returned %ld\n", hRes);
        if (hRes <= (HINSTANCE)32)
        {
            CString strError;
            strError.LoadString(IDS_ERROR_BROWSER);
            strError += szFile;
            AfxMessageBox(strError);
            return NULL;
        }

        CString strCmd = _T("WRITE ") + strFile;
        BOOL bRes = ::CreateProcess("C:\\WINDOWS\\NOTEPAD.EXE",
strCmd.GetBuffer(128), NULL, NULL,
FALSE, NULL, NULL, NULL, &si, &pi);
        if (!bRes)

```

```

        return NULL;
        ::CloseHandle(pi.hThread);
        ::WaitForInputIdle(pi.hProcess, 10000);
        ::CloseHandle(pi.hProcess);
        int nRep=0;
        while (!hwndFind && (nRep++ < 20))
        {
            ::Sleep(1000);
            TRACE("z");
            hwndFind = ::FindWindow("Notepad", NULL);    //
            "Tortoise's Experiment"
        }
    }
    // ::SendMessage(hwndFind, WM_SETTEXT, 0, (LPARAM)(LPCSTR)strFile);
    return hwndFind;
}

// 1. Find a running Notepad (or initiate a new Notepad) application.
// 2. Set the contents in the notepad (Edit control) with some texts.
void CMainFrame::SendTextToNotepad()
{
    CEdit* pE = m_wndHistoryPanel.GetEditHistory();
    if (!pE)
        return;
    CString strText;
    pE->GetWindowText(strText);
    int len = strText.GetLength();
    TRACE1("SendTextToNotepad - length:%d\n", len);
    if (len == 0)
        return;
    LPCSTR szText = strText;
    HWND hwndFind = GetNotepad();
    HWND hwndChild = ::GetWindow(hwndFind, GW_CHILD);
    HWND hwnd = hwndChild;
    UINT msg = WM_SETTEXT;
    WPARAM wParam = 0;
    LPARAM lParam = (LPARAM)szText;
    LRESULT res = ::SendMessage(hwnd, msg, wParam,
    lParam);
    TRACE("::SendMessage(0x%x, msg, wParam, lParam); returned %lx\n",
        hwnd, msg, wParam, lParam, res);
}

void CMainFrame::OnEditCopy()
{
    CEdit* pEdit = GetEditFocus();
    if (pEdit)
        pEdit->Copy();
}

void CMainFrame::OnEditCut()
{
    CEdit* pEdit = GetEditFocus();
    if (pEdit)
        pEdit->Cut();
}

```

```

void CMainFrame::OnEditNotepad()
{
    //      CEditHistory* pEH = m_wndHistoryPanel.GetEditHistory();
    //      if (pEH)
    //          pEH->SendTextToNotepad();
    SendTextToNotepad();
}

void CMainFrame::OnEditPaste()
{
    CEdit* pEdit = GetEditFocus();
    if (pEdit)
        pEdit->Paste();
}

void CMainFrame::OnEditUndo()
{
    CEdit* pEdit = GetEditFocus();
    if (pEdit)
        pEdit->Undo();
}

void CMainFrame::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
    CEdit* pEdit = GetEditFocus();
    if (pEdit)
    {
        int n1, n2;
        pEdit->GetSel(n1, n2);
        pCmdUI->Enable(n1 != n2);      // IsTextSelected
    }
    else
    {
        pCmdUI->Enable(FALSE);
    }
}

void CMainFrame::OnUpdateEditCut(CCmdUI* pCmdUI)
{
    CEdit* pEdit = GetEditFocus();
    if (pEdit)
    {
        int n1, n2;
        pEdit->GetSel(n1, n2);
        pCmdUI->Enable(n1 != n2);      // IsTextSelected
    }
    else
    {
        pCmdUI->Enable(FALSE);
    }
}

void CMainFrame::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(::IsClipboardFormatAvailable(CF_TEXT));
}

```

```

void CMainFrame::OnUpdateEditUndo(CCmdUI* pCmdUI)
{
    CEdit* pEdit = GetEditFocus();
    pCmdUI->Enable(pEdit ? pEdit->CanUndo() : FALSE);
}

// Get current (focused) edit control
CEdit* CMainFrame::GetEditFocus()
{
    CWnd* pFocus = GetFocus();
    CEdit* pEditSend = m_wndControlPanel.GetEditSend();
    CEdit* pEditHistory = m_wndHistoryPanel.GetEditHistory();
    if (pEditSend == pFocus)
        return pEditSend;
    if (pEditHistory == pFocus)
        return pEditHistory;
    return NULL;
}

BOOL CMainFrame::ShellBrowseURL(LPCTSTR szURL)
{
    TRACE("Browsing %s\n", szURL);
    HINSTANCE hRes = ::ShellExecute(GetSafeHwnd(), "open", szURL, NULL,
    NULL, SW_SHOWNORMAL);
    TRACE("::ShellExecute(%xh) returned %ld\n", hRes);
    if (hRes <= (HINSTANCE)32)
    {
        CString strError;
        strError.LoadString(IDS_ERROR_BROWSER);
        strError += szURL;
        AfxMessageBox(strError);
        return FALSE;
    }
    return TRUE;
}

/*
BOOL CMainFrame::LaunchBrowser(LPCTSTR szURL)
{
    HKEY hKey;
    CString strBuf;
    LONG cbSize = 255;
    if (::RegOpenKey(HKEY_LOCAL_MACHINE,
        "Software\\Classes\\http\\shell\\open", &hKey))
    {
        TRACE("ERROR Open Registry\n");
        return FALSE;
    }

    // "C:\PROGRA~1\PLUS!\MICROS~1\iexplore.exe" -nohome
    ::RegQueryValue(hKey, "Command", strBuf.GetBuffer(cbSize), &cbSize);
    ::RegCloseKey(hKey);
    strBuf.ReleaseBuffer();

    CString strExec;
    CString strURL(szURL);
    int nTot = strBuf.GetLength();
    int sep = strBuf.ReverseFind(' ');

```

```

        int nRead = nTot - sep;
        strExec = strBuf.Left(nTot-nRead) + " " + strURL;
        TRACE("::WinExec(%s)\n", strExec);
        ::WinExec(strExec, SW_SHOWNORMAL);

        return TRUE;
    }
    */

void CMainFrame::OnHelpHomepage()
{
    CString strURL;
    if (strURL.LoadString(IDS_UNICHAT_HOMEURL))
        ShellBrowseURL(strURL);
}

BOOL CMainFrame::Is256Palette() const
{
    BOOL bResult=TRUE;
    // Get a screen DC to work with.
    HWND hwndActive = ::GetActiveWindow();
    HDC hdcScreen = ::GetDC(hwndActive);
    ASSERT(hdcScreen);

    // Make sure we are on a palettized device.
    if (!(::GetDeviceCaps(hdcScreen, RASTERCAPS) & RC_PALETTE))
    {
        bResult = FALSE;
    }
    else
    {
        // Get the number of system colors and the number of palette
        // entries. Note that on a palletized device the number of
        // colors is the number of guaranteed colors, i.e., the number
        // of reserved system colors.
        int iSysColors = ::GetDeviceCaps(hdcScreen, NUMCOLORS);
        int iPalEntries = ::GetDeviceCaps(hdcScreen, SIZEPALETTE);

        // If there are more than 256 colors we are wasting our time.
        if (iSysColors < 0 || iSysColors > 256)
        {
            bResult = FALSE;
        }
    }
    ::ReleaseDC(hwndActive, hdcScreen);
    return bResult;
}

////////////////////////////////////
// ChatSock Message Handler
LRESULT CMainFrame::OnCsAddChannel(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsAddChannel(wParam, lParam);
}

LRESULT CMainFrame::OnCsPrivateMsg(WPARAM wParam, LPARAM lParam)

```

```

{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsPrivateMsg(wParam, lParam);
}

LRESULT CMainFrame::OnCsQueryData(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsQueryData(wParam, lParam);
}

LRESULT CMainFrame::OnCsInvite(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsInvite(wParam, lParam);
}

LRESULT CMainFrame::OnCsGotMemList(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsGotMemList(wParam, lParam);
}

LRESULT CMainFrame::OnCsAddMember(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsAddMember(wParam, lParam);
}

LRESULT CMainFrame::OnCsDelMember(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsDelMember(wParam, lParam);
}

LRESULT CMainFrame::OnCsDelChannel(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsDelChannel(wParam, lParam);
}

LRESULT CMainFrame::OnCsModeMember(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsModeMember(wParam, lParam);
}

LRESULT CMainFrame::OnCsModeChannel(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsModeChannel(wParam, lParam);
}

LRESULT CMainFrame::OnCsTextA(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsTextA(wParam, lParam);
}

```

```

}

LRESULT CMainFrame::OnCsData(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsData(wParam, lParam);
}

LRESULT CMainFrame::OnCsWhisperText(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsWhisperText(wParam, lParam);
}

LRESULT CMainFrame::OnCsWhisperData(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsWhisperData(wParam, lParam);
}

LRESULT CMainFrame::OnCsNewTopic(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsNewTopic(wParam, lParam);
}

LRESULT CMainFrame::OnCsNewNick(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsNewNick(wParam, lParam);
}

LRESULT CMainFrame::OnChannelFullRetry(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnChannelFullRetry(wParam, lParam);
}

void CMainFrame::OnUpdateDate(CCmdUI* pCmdUI)
{
    // CG: This function was inserted by 'Status Bar' component.

    // Get current date and format it
    CTime time = CTime::GetCurrentTime();
    CString strDate = time.Format(_T("%x"));

    // BLOCK: compute the width of the date string
    CSize size;
    {
        HGDIOBJ hOldFont = NULL;
        HFONT hFont = (HFONT)m_wndStatusBar.SendMessage(WM_GETFONT);
        CClientDC dc(NULL);
        if (hFont != NULL)
            hOldFont = dc.SelectObject(hFont);
        size = dc.GetTextExtent(strDate);
        if (hOldFont != NULL)
            dc.SelectObject(hOldFont);
    }
}

```

```

    }

    // Update the pane to reflect the current date
    UINT nID, nStyle;
    int nWidth;
    m_wndStatusBar.GetPaneInfo(m_nDatePaneNo, nID, nStyle, nWidth);
    m_wndStatusBar.SetPaneInfo(m_nDatePaneNo, nID, nStyle, size.cx);
    pCmdUI->SetText(strDate);
    pCmdUI->Enable(TRUE);
}

void CMainFrame::OnUpdateTime(CCmdUI*, pCmdUI)
{
    // CG: This function was inserted by 'Status Bar' component.

    // Get current date and format it
    CTime time = CTime::GetCurrentTime();
    CString strTime = time.Format(_T("%X"));

    // BLOCK: compute the width of the date string
    CSize size;
    {
        HGDIOBJ hOldFont = NULL;
        HFONT hFont = (HFONT)m_wndStatusBar.SendMessage(WM_GETFONT);
        CClientDC dc(NULL);
        if (hFont != NULL)
            hOldFont = dc.SelectObject(hFont);
        size = dc.GetTextExtent(strTime);
        if (hOldFont != NULL)
            dc.SelectObject(hOldFont);
    }

    // Update the pane to reflect the current time
    UINT nID, nStyle;
    int nWidth;
    m_wndStatusBar.GetPaneInfo(m_nTimePaneNo, nID, nStyle, nWidth);
    m_wndStatusBar.SetPaneInfo(m_nTimePaneNo, nID, nStyle, size.cx);
    pCmdUI->SetText(strTime);
    pCmdUI->Enable(TRUE);
}

BOOL CMainFrame::InitStatusBar(UINT *pIndicators, int nSize, int nSeconds)
{
    // CG: This function was inserted by 'Status Bar' component.

    // Create an index for the DATE pane
    m_nDatePaneNo = nSize++;
    pIndicators[m_nDatePaneNo] = ID_INDICATOR_DATE;
    // Create an index for the TIME pane
    m_nTimePaneNo = nSize++;
    nSeconds = 1;
    pIndicators[m_nTimePaneNo] = ID_INDICATOR_TIME;

    // TODO: Select an appropriate time interval for updating
    // the status bar when idling.
    m_wndStatusBar.SetTimer(0x1000, nSeconds * 1000, NULL);
}

```



```

        return m_wndStatusBar.SetIndicators(pIndicators, nSize);
    }

// SATYA CHANGES START
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpCS, CCreateContext*
pContext)
{
    TRACE("\nCMainFrame::OnCreateClient\n");

    VERIFY(m_wndSplitter.CreateStatic(this, 2, 1));

    TRACE("\nCMainFrame::OnCreateClient View1\n");

    VERIFY(m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CWebView),
        CSize(60,60), pContext));

    VERIFY(m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CUC2View),
        CSize(480, 480), pContext));
    TRACE("\nCMainFrame::OnCreateClient View2\n");
    m_wndSplitter.SetActivePane(1,0,NULL);
    // return CFrameWnd::OnCreateClient(lpCS, pContext);
    return TRUE;
}

void CMainFrame::OnSysCommand( UINT nID, LPARAM lParam )
{
    // When minimise the Unichat application Window
    // it should hide the application i.e it should not show on Status bar
    if (nID == SC_MINIMIZE)
    {
        // Call OnPopMinimise to the Work
        OnPopMinimise();
    }
    else
    {
        CFrameWnd ::OnSysCommand(nID,lParam);
    }
}

LRESULT CMainFrame::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        // The following code will add the unichat icon on the status
area
        // and process its notification messages from the Popupmenu.
        case WM_UCICON_NOTIFY:
            // If user clicks on the icon show the Window if it is
            // hidden

            if(lParam == WM_LBUTTONDOWNCLK)
            {

```

```

//CHANGES_MADE_FOR_UNICHAT_2
CUC2Doc *pDoc = (CUC2Doc*) this->GetActiveDocument();

if (pDoc->m_bIsJoinChannelVisible)
{
    MessageBeep(C_MAX_BEEP_TIME);
    return 0;
}
if (this->IsWindowVisible())
{
    OnPopMinimise();
}
else
{
    OnPopMaximise();
}
break;
}
if (lParam == WM_RBUTTONDOWN)
{
    CUC2Doc *pDoc = (CUC2Doc*) this->GetActiveDocument();
    CMenu PopUpMenu;
    PopUpMenu.CreatePopupMenu();
    if (this->IsWindowVisible())
    {
        PopUpMenu.AppendMenu(MF_ENABLED|MF_STRING, UCMN_UNICHAT, "UniChat");
        //CHANGES_MADE_FOR_UNICHAT_2
        if (!pDoc->m_bIsJoinChannelVisible)
        {
            PopUpMenu.AppendMenu(MF_SEPARATOR);
        }
        PopUpMenu.AppendMenu(MF_ENABLED|MF_STRING, UCMN_MINIMISE, "Minimise");
        PopUpMenu.AppendMenu(MF_SEPARATOR);

        PopUpMenu.AppendMenu(MF_ENABLED|MF_STRING, UCMN_EXIT, "Exit");
    }
    else
    {
        PopUpMenu.AppendMenu(MF_ENABLED|MF_STRING, UCMN_UNICHAT, "UniChat");
        if (!pDoc->m_bIsJoinChannelVisible)
        {
            PopUpMenu.AppendMenu(MF_SEPARATOR);
        }
        PopUpMenu.AppendMenu(MF_GRAYED|MF_STRING, UCMN_MAXIMISE, "Maximise");
        PopUpMenu.AppendMenu(MF_SEPARATOR);
        PopUpMenu.AppendMenu(MF_ENABLED|MF_STRING, UCMN_EXIT, "Exit");
    }
}

```

```

        PopUpMenu.TrackPopupMenu(TPM_LEFTALIGN, 900, 900, this, NULL);

    }
    break;

default:
    return CFrameWnd::WindowProc(message, wParam, lParam);
}

return CFrameWnd::WindowProc(message, wParam, lParam);
}

// Function Name : AddUCIconOnStatusArea
// Parameters      : NONE
// Purpose         : This function adds an unichat Icon on the Status area.
// Return value    : TRUE : if it successfully adds an Icon. Otherwise FALSE.
BOOL CMainFrame:: AddUCIconOnStatusArea()
{
    NOTIFYICONDATA UCIconData;
    HICON hUCIcon = NULL;
    LPSTR lpszUCTip = "UniChat";

    // Load the icon from the Resources
    hUCIcon = LoadIcon(AfxGetInstanceHandle(), MAKEINTRESOURCE(IDI_UCICON));
    ASSERT(hUCIcon != NULL);
    UCIconData.cbSize = sizeof(NOTIFYICONDATA);
    UCIconData.hWnd = this->m_hWnd;
    UCIconData.uID = ID_UCICON;
    UCIconData.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP;
    UCIconData.uCallbackMessage = WM_UCICON_NOTIFY;
    UCIconData.hIcon = hUCIcon;
    lstrcpy(UCIconData.szTip, lpszUCTip, sizeof(UCIconData.szTip));
    BOOL Result = Shell_NotifyIcon(NIM_ADD, &UCIconData);
    return Result;
}

// Function Name : DeleteUCIconOnStatusArea
// Parameters      : NONE
// Purpose         : This function deletes the Icon from the Status area
// Return value    : TRUE : if it successfully deletes Icon. Otherwise FALSE.
BOOL CMainFrame :: DeleteUCIconOnStatusArea()
{
    NOTIFYICONDATA UCIconData;
    UCIconData.cbSize = sizeof(NOTIFYICONDATA);
    UCIconData.hWnd = this->m_hWnd;
    UCIconData.uID = ID_UCICON;

    BOOL Result = Shell_NotifyIcon(NIM_DELETE, &UCIconData);
    return Result;
}

// Message Handler for the Minimise Popup menu
void CMainFrame:: OnPopMinimise()
{
    if(this->IsWindowVisible())
    {
        ShowWindow(SW_HIDE);
    }
}

```

```

        CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
        pDoc->QuitChannel();
    }
}
// Message Handler for Maximise Popup menu
void CMainFrame:: OnPopMaximise()
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();

    if(!this->IsWindowVisible())
    {
        ShowWindow(SW_SHOW);
    }
    //if the user is in the channel then
    pDoc->OnBtnCreate();
}

// Message Handler for Unichat PopUp menu
void CMainFrame:: OnUniChat()
{
    CString strURL;
    if (strURL.LoadString(IDS_UNICHAT_HOMEURL))
        ShellBrowseURL(strURL);
}
// Message Handler for Exit PopUp menu
void CMainFrame :: OnUCExit()
{
    this->SendMessage(WM_CLOSE,0,0L);
}
// SATYA CHANGES END

```

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT NETWORKS INC
//=====

#if !defined(AFX_MAINFRM_H__A131386D_A610_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_MAINFRM_H__A131386D_A610_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "UC2Panel.h"
#include "UC2History.h"
#include "UCSplitterWnd.h" // for CUCSplitterWnd

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
    CPalette*          GetPalette()          { return m_pPalette; }
    CUC2Panel*         GetPanel()            { return &m_wndControlPanel; }
}

    CUC2History*       GetHistoryPanel() { return &m_wndHistoryPanel; }
    CTime              m_tmStart;

// Operations
public:
    CPalette*          SetPalette(CPalette* pPalette);
    void               SetFreeze(const BOOL bFreeze=TRUE);
    void               AdjustFrame(const int nW, const int nH);
    BOOL               InitControlPanel(CPalette* pPal);
    void               ShowHistoryPanel(const BOOL bShow=TRUE);
    BOOL               ShellBrowseURL(LPCTSTR szURL);
    BOOL               Is256Palette() const;
    void               SetAskBeforeClose(const BOOL bABC) { m_bAskBeforeClose =
bABC; }

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
    //}}AFX_VIRTUAL

// Implementation
public:
    int m_nDatePaneNo;
    int m_nTimePaneNo;
```

```

        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    // CToolBar m_wndToolBar;
    CUC2Panel m_wndControlPanel;
    CUC2History m_wndHistoryPanel;
    // Splitter window which adds Banner Window
    CUCSplitterWnd m_wndSplitter;
    // SATYA CHANGES START
    // Added functions for Unicaht Icon
    BOOL AddUCIconOnStatusArea(); // This function adds an Icon on the
Desktop
    BOOL DeleteUCIconOnStatusArea(); // This function deletes the icon from
Desktop
    // SATYA CHAGES END

// Generated message map functions
protected:
    afx_msg void OnUpdateDate(CCmdUI* pCmdUI);
    afx_msg void OnUpdateTime(CCmdUI* pCmdUI);
    CEdit* GetEditFocus();
    HWND GetNotepad();
    void SendTextToNotepad();

    CPalette* m_pPalette;
    BOOL m_bFreeze;
    CRect m_rcView;
    CRect m_rcMax;
    BOOL m_bAskBeforeClose;
    // Added windProc to trap the Messages
    virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);

    //{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnClose();
    afx_msg void OnGetMinMaxInfo(MINMAXINFO FAR* lpMMI);
    afx_msg BOOL OnQueryNewPalette();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    afx_msg void OnDestroy();
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnEditCopy();
    afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
    afx_msg void OnEditCut();
    afx_msg void OnUpdateEditCut(CCmdUI* pCmdUI);
    afx_msg void OnEditNotepad();
    afx_msg void OnEditPaste();
    afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
    afx_msg void OnEditUndo();
    afx_msg void OnUpdateEditUndo(CCmdUI* pCmdUI);
    afx_msg void OnHelpHomepage();
    // SATYA CHANGES START
    // Following are the Message Handlers for Popup Menu

```

```

afx_msg void OnPopMinimise();
afx_msg void OnPopMaximise();
afx_msg void OnUniChat();
afx_msg void OnUCExit();
//SATYA CHANGES END

//}}AFX_MSG
// ChatSock
afx_msg LRESULT OnCsAddChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsPrivateMsg(WPARAM, LPARAM);
afx_msg LRESULT OnCsQueryData(WPARAM, LPARAM);
afx_msg LRESULT OnCsInvite(WPARAM, LPARAM);
afx_msg LRESULT OnCsGotMemList(WPARAM, LPARAM);
afx_msg LRESULT OnCsAddMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsDelMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsDelChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsModeMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsModeChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsTextA(WPARAM, LPARAM);
afx_msg LRESULT OnCsData(WPARAM, LPARAM);
afx_msg LRESULT OnCsWhisperText(WPARAM, LPARAM);
afx_msg LRESULT OnCsWhisperData(WPARAM, LPARAM);
afx_msg LRESULT OnCsNewTopic(WPARAM, LPARAM);
afx_msg LRESULT OnCsNewNick(WPARAM, LPARAM);
afx_msg LRESULT OnChannelFullRetry(WPARAM, LPARAM);
afx_msg void OnSysCommand( UINT nID, LPARAM lParam );
DECLARE_MESSAGE_MAP()

private:
    BOOL InitStatusBar(UINT *pIndicators, int nSize, int nSeconds);
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_MAINFRM_H__A131386D_A610_11D1_80E2_080009B9F339__INCLUDED_

```

masterJasc.pal

JASC-PAL

0100

256

0 0 0

128 0 0

0 128 0

128 128 0

0 0 128

128 0 128

0 128 128

192 192 192

192 220 192

166 202 240

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

255 255 255

131 231 131

91 91 111

107 0 27

199 27 51

255 63 83

255 115 111

39 47 87

0 103 179

47 159 235

147 211 255

55 7 0

119 47 11

187 115 43

255 203 87

35 47 23

71 95 47

111 147 71

151 199 95

119 47 11

187 115 43

231 195 151

255 235 187

23 35 107

27 75 155

31 135 203

31 215 255

115 47 11

187 115 43

247 175 0

255 231 0

35 47 23

71 95 47

111 147 71

151 199 95

139 0 31

223 51 63

masterJasc.pal

239 79 83
255 119 111
47 35 19
103 95 67
163 163 143
203 203 187
255 215 183
247 187 147
243 167 91
239 175 139
0 7 0
0 31 0
0 55 0
7 79 7
11 103 11
19 131 19
31 155 27
43 179 39
55 203 51
71 219 67
91 235 87
115 243 107
143 255 135
167 255 163
195 255 191
223 255 219
55 7 0
103 35 0
155 87 0
203 159 0
255 255 0
255 255 51
255 255 107
255 255 159
255 255 215
8 40 32
23 71 51
35 99 67
51 127 83
71 155 95
95 183 111
123 215 127
128 0 128
143 15 143
159 31 159
175 55 175
191 79 191
207 107 207
223 139 223
239 175 239
255 215 255
207 103 31
211 155 119
175 115 0
195 131 35
215 155 75
235 183 127

masterJasc.pal

255 219 187
119 0 51
163 0 47
207 0 27
255 0 0
255 91 71
255 167 143
255 231 219
39 23 183
39 35 183
39 35 203
47 35 219
71 55 223
83 75 223
95 91 223
0 0 255
0 99 219
15 155 247
67 219 255
139 255 255
199 255 255
231 255 255
135 127 235
136 24 16
224 104 104
24 48 80
72 104 160
24 80 72
96 136 96
56 24 16
40 24 16
48 24 8
59 31 0
223 107 27
239 131 63
247 163 107
32 32 24
8 0 0
72 16 8
64 56 32
48 48 48
40 40 32
56 56 48
48 48 40
56 48 40
48 40 24
80 48 8
64 40 16
88 72 32
96 64 32
120 72 40
104 56 8
120 64 16
128 80 24
120 88 64
144 96 64
167 95 55

masterJasc.pal

176 104 56
176 104 16
152 96 16
183 123 87
199 123 99
192 144 80
184 128 16
200 136 56
224 136 64
216 144 32
224 160 56
240 160 80
248 176 72
248 184 120
192 160 104
200 184 112
8 40 32
24 40 8
32 48 24
24 64 0
40 80 8
56 80 40
64 80 0
72 104 40
56 104 16
88 112 64
96 112 32
112 128 24
96 136 64
40 56 16
64 80 24
88 96 24
96 128 0
120 136 24
88 152 40
144 152 40
136 168 8
128 160 72
152 176 40
176 192 48
200 216 56
216 224 64
232 240 72
248 248 104
64 64 56
72 64 56
72 72 64
88 80 72
88 88 80
96 96 88
112 104 88
104 104 96
112 112 104
120 112 104
128 120 112
136 128 80
144 128 112

masterJasc.pal

152 144 136
136 136 128
160 160 144
160 152 128
160 168 168
144 152 112
168 168 152
176 176 168
176 160 136
184 168 144
192 176 152
200 184 160
200 192 176
200 192 184
224 200 168
208 208 192
224 208 192
224 224 216
240 232 200
248 240 224
248 248 248
255 251 240
160 160 164
128 128 128
255 0 0
0 255 0
255 255 0
0 0 255
255 0 255
0 255 255
255 255 255

MemberInfo.cpp

```
// MemberInfo.cpp : implementation of the classes for protocol
//

#include "stdafx.h"
#include "MemberInfo.h"
#include "Parser.h"
#include "Actor.h"
#include "Behavior.h"
#ifdef MAPEDITOR
#include "UC2.h"
#include "MainFrm.h" // GetUserID
#endif

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CParser gParser;

////////////////////////////////////
// CMemberInfo
CMemberInfo::CMemberInfo()
{
    m_pMember        = NULL;
    m_nCharID         = 0;
    m_nSex            = SEX_MAN;
    m_nAge            = 20;
    m_nBubbleKind     = m_nSex;
    m_nVersion        = CLIENT_VERSION;
    m_nBehavior       = AB_STANDINGF;
    m_wState          = AS_STAND | DA_FR;
    m_ptTID           = CPoint(5,11);
}

CMemberInfo::~CMemberInfo()
{
    if (m_pMember)
        m_pMember->Release();
}

CMemberInfo& CMemberInfo::operator=(const CMemberInfo& mi) // mi from
{
    if (this == &mi)
        return *this;

    if (m_pMember)
        m_pMember->Release();
    m_pMember        = mi.GetMember(); // NULL for actors in script mode
    m_nVersion        = mi.GetVersion();
    m_nCharID         = mi.GetCharID();
    m_nSex            = mi.GetSex();
    m_nAge            = mi.GetAge();
    m_nBubbleKind     = mi.GetBubbleKind();
    m_strNick         = *mi.GetNick();
    m_strRealName     = *mi.GetRealName();
    m_strProfile      = *mi.GetProfile();
}
```

```

        m_strUserID          = *mi.GetUserID();
        m_strHyperlink       = *mi.GetHyperlink();
        m_nBehavior          = mi.GetBehavior();
        m_wState              = mi.GetState();
        m_ptTID               = mi.GetTileID();
        return *this;
    }

CMemberInfo::CMemberInfo(const CMemberInfo& mi) // Copy Constructor
{
    *this = mi;
}

BOOL CMemberInfo::SetMember(PICS_MEMBER pMember)
{
    if (m_pMember)
    {
        m_pMember->Release();
    }
    m_pMember = pMember;
    if (m_pMember)
    {
        m_pMember->AddRef();
        BYTE* pb;
        BOOL fAnsi;
        HRESULT hr = m_pMember->HrGetName(&pb, &fAnsi);
        if (!FAILED(hr) && fAnsi)
            m_strNick = (char*)pb;
        return TRUE;
    }
    return FALSE;
}

const PICS_MEMBER CMemberInfo::GetMember() const
{
    if (m_pMember)
        m_pMember->AddRef();
    return m_pMember;
}

// Set this member info from the specified string
BOOL CMemberInfo::SetMemberInfo(const char* src)
{
    //`nActorID`nBubbleKind`strHandle`strRealName`strProfile`ptBase`nHDir`strUser
    ID(`)
    gParser.CopyBuffer(src);
    if (!gParser.GetValueRightToken(m_nVersion, UC2TOKEN))
        return FALSE;
    if (!gParser.GetValueRightToken(m_nCharID, UC2TOKEN))
        return FALSE;
    if (!gParser.GetValueRightToken(m_nSex, UC2TOKEN))
        return FALSE;
    if (!gParser.GetValueRightToken(m_nAge, UC2TOKEN))
        return FALSE;
    if (!gParser.GetValueRightToken(m_nBubbleKind, UC2TOKEN)) return
FALSE;

```

```

        if (!gParser.GetValueRightToken(m_strNick, UC2TOKEN))
            return FALSE;
        if (!gParser.GetValueRightToken(m_strRealName, UC2TOKEN)) return
FALSE;
        if (!gParser.GetValueRightToken(m_strProfile, UC2TOKEN)) return
FALSE;
        if (!gParser.GetValueRightToken(m_strUserID, UC2TOKEN)) return
FALSE;
        if (!gParser.GetValueRightToken(m_strHyperlink, UC2TOKEN)) return
FALSE;
        if (!gParser.GetValueRightToken(m_nBehavior, UC2TOKEN)) return
FALSE;
        if (!gParser.GetValueRightToken(m_wState, UC2TOKEN)) return
FALSE;
        if (!gParser.GetValueRightToken(m_ptTID, UC2TOKEN)) return
FALSE;
        return TRUE;
    }

// Get this member info to the specified string
void CMemberInfo::GetMemberInfo(CString& str) const
{
    str.Format("%d`%d`%d`%d`%d,`%s`%s`%s`%s`%s`%d`%d`(%d,%d)`",
        m_nVersion, m_nCharID, m_nSex, m_nAge, m_nBubbleKind,
        m_strNick, m_strRealName, m_strProfile, m_strUserID,
        m_strHyperlink,
        m_nBehavior, m_wState, m_ptTID.x, m_ptTID.y);
}

void CMemberInfo::SetNick(LPCSTR szNick)
{
    m_strNick = szNick;
    if (m_strNick.GetLength() > 20)
        m_strNick.ReleaseBuffer(20);
}

void CMemberInfo::SetRealName(LPCSTR szRN)
{
    m_strRealName = szRN;
    if (m_strRealName.GetLength() > 20)
        m_strRealName.ReleaseBuffer(20);
}

void CMemberInfo::SetProfile(LPCSTR szProf)
{
    m_strProfile = szProf;
    if (m_strProfile.GetLength() > 256)
        m_strProfile.ReleaseBuffer(256);
}

#ifdef MAPEDITOR
// Load my member information from the system registry
BOOL CMemberInfo::LoadMyInfo()
{
    #ifdef _UNITEL
    // m_strRealName = *((CMainFrame*)AfxGetMainWnd())->GetUserName();
    #endif // _UNITEL
}

```

MemberInfo.cpp

```
//      m_strUserID = *((CMainFrame*)AfxGetMainWnd())->GetUserID();
      return ((CUC2App*)AfxGetApp())->RegGetMemberInfo(*this);
}

BOOL CMemberInfo::SaveMyInfo()
{
    return ((CUC2App*)AfxGetApp())->RegSetMemberInfo(*this);
}
#endif // MAPEDITOR
```


MemberInfo.h

```
//
//  CMemberInfo
//
//  (C) Programmed by Kim,
//  UNICHAT Lab
//  Information Technology Institue
//
//
/////////////////////////////////////////////////////////////////

#ifndef __MEMBERINFO_H_
#define __MEMBERINFO_H_

class CMemberInfo // : public CObject
{
public:
    CMemberInfo();
    CMemberInfo(const CMemberInfo& mi); // Copy Constructor
    virtual ~CMemberInfo();
    CMemberInfo& operator=(const CMemberInfo& mi);

// Attributes
    const PICS_MEMBER GetMember() const;
    const PICS_MEMBER RefMember() const { return m_pMember; } // Just
reference
    void                GetMemberInfo(CString& str) const;
    int                 GetVersion() const                { return
m_nVersion; }
    int                 GetCharID() const                 { return m_nCharID; }
    int                 GetSex() const                    { return m_nSex; }
}
    int                 GetAge() const                    { return m_nAge; }
}
    int                 GetBubbleKind() const             { return m_nBubbleKind; }
}
    const CString*      GetNick() const                   { return &m_strNick; }
    const CString*      GetRealName() const               { return
&m_strRealName; }
    const CString*      GetProfile() const                { return &m_strProfile; }
}
    const CString*      GetUserID() const                 { return &m_strUserID; }
    const CString*      GetHyperlink() const              { return &m_strHyperlink; }
    BOOL                HasHyperlink() const              { return
(!m_strHyperlink.IsEmpty()); }
    int                 GetBehavior() const                { return
m_nBehavior; }
    WORD                GetState() const                   { return m_wState; }
    CPoint              GetTileID() const                  { return m_ptTID; }

// Operations
    // For script-created actor object, set pMember=NULL, and assign a
handle
    BOOL                SetMember(PICS_MEMBER pMember);
    BOOL                SetMemberInfo(const char* src); // set member data from
string
    void                SetVersion(const int nVer)         { m_nVersion = nVer; }
    void                SetCharID(const int nID)           { m_nCharID = nID; }
```

MemberInfo.h

```

void SetSex(const int n) { m_nSex = n; }
void SetAge(const int n) { m_nAge = n; }
void SetBubbleKind(const int n) { m_nBubbleKind = n; }
void SetNick(LPCSTR szNick);
void SetRealName(LPCSTR szRN);
void SetProfile(LPCSTR szProf);
void SetUserID(LPCSTR szUID) { m_strUserID = szUID; }
void SetHyperlink(LPCSTR szHL) { m_strHyperlink = szHL; }
void SetBehavior(const int beh) { m_nBehavior = beh; }
void SetState(const WORD wS) { m_wState = wS; }
void SetSA(const WORD wSA) { m_wState = (wSA | (m_wState
& ~AS_MASK)); }
void SetDA(const WORD wDA) { m_wState = (wDA | (m_wState
& ~DA_MASK)); }
void SetTileID(const CPoint& pt) { m_ptTID = pt; }
#endifdef MAPEDITOR
    BOOL LoadMyInfo();
    BOOL SaveMyInfo();
#endif

protected:
    enum SEX
    {
        SEX_MAN,
        SEX_WOMAN
    };
    PICS_MEMBER m_pMember; // NULL for actors in script mode
    int m_nVersion; // Client Version of this
member
    int m_nCharID; // Actor ID, 0,1,..., the
kind of actors
    int m_nSex; // 0 man, 1 woman
    int m_nAge;
    int m_nBubbleKind;
    CString m_strNick; // Chat ID
    CString m_strRealName;
    CString m_strProfile;
    CString m_strUserID;
    CString m_strHyperlink;
    int m_nBehavior;
    WORD m_wState; // Actor State and Direction
Attribute
    CPoint m_ptTID; // Tile ID the actor stands
on
    // int m_nElev;
};

#endif

```

MemberListDlg.cpp

```
// MemberListDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "MemberListDlg.h"
#include "UC2Doc.h"
#include "UC2CS.h"
#include "PPChannel.h"
#include "Actor.h"
#include "Parser.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CMemberListDlg dialog

CMemberListDlg::CMemberListDlg(CWnd* pParent /*=NULL*/)
: CDialog(CMemberListDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CMemberListDlg)
    m_strCount = _T("");
    m_strMessage = _T("");
    //}}AFX_DATA_INIT
    m_pDoc = NULL;
    //
    m_dwChannelID = 0L;
    m_font.CreateFont(-12, 0, 0, 0, FW_BOLD,
        FALSE, FALSE, 0, // bItalic, bUnderline, cStrikeOut
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH || FF_DONTCARE,
#ifdef _KOREAN
        "t4.2A4");
#else
        "Times New Roman");
#endif
    m_ilMember.Create(IDB_IL_MEMBER, 16, 1, CLR_NONE);
    m_nIndex = -1;
    m_strChannel.Empty();
}

void CMemberListDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMemberListDlg)
    DDX_Control(pDX, IDC_LIST_MEMBER, m_lcMember);
    DDX_Text(pDX, IDC_STATIC_MEMBER_COUNT, m_strCount);
    DDX_Text(pDX, IDC_STATIC_MESSAGE, m_strMessage);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMemberListDlg, CDialog)
    //{{AFX_MSG_MAP(CMemberListDlg)
```

```

ON_BN_CLICKED(IDC_BTN_INVITE, OnBtnInvite)
ON_BN_CLICKED(IDC_BTN_RENEW, OnBtnRenew)
ON_NOTIFY(NM_CLICK, IDC_LIST_MEMBER, OnClickListMember)
ON_NOTIFY(LVN_DELETEITEM, IDC_LIST_MEMBER, OnDeleteitemListMember)
ON_NOTIFY(NM_DBLCLK, IDC_LIST_MEMBER, OnDblclkListMember)
ON_WM_DESTROY()
//}}AFX_MSG_MAP
ON_MESSAGE(CMD_QUERY_MEMBERS, OnQueryMembers)
ON_MESSAGE(CMD_QUERY_MEMBERS_END, OnQueryMembersEnd)
ON_MESSAGE(CMD_QUERY_NOMATCHES, OnQueryNoMatches)
ON_MESSAGE(CMD_QUERY_GET_REALNAME, OnQueryGetRealname)
END_MESSAGE_MAP()

////////////////////////////////////
// CMemberListDlg message handlers

void CMemberListDlg::OnBtnInvite()
{
    if (m_nIndex == -1)
    {
        AfxMessageBox(IDS_INVITE_SELECT_MEMBER);
        return;
    }
    CString strNick = m_lcMember.GetItemText(m_nIndex, 0); //
    SubItem=0
    CUC2Channel* pChan = m_pDoc->GetChannel();
    if (!pChan)
    {
        AfxMessageBox(IDS_INVITE_OPEN_CHANNEL);
        return;
    }

    CString strMsg;
    CActor* pA = m_pDoc->GetThisActor();
    ASSERT(pA);
    if (*pA->GetNick() == strNick)
    {
        if (strMsg.LoadString(IDS_INVITE_ERROR_MYSELF))
            AfxMessageBox(strMsg);
        return;
    }

    int len = strNick.GetLength();
    CS_INVITEINFO csiInfo;
    csiInfo.dwUserID = 0;
    csiInfo.pvNickTo = strNick.GetBuffer(len);
    if (pChan->FSendInvite(&csiInfo))
    {
        AfxFormatString1(strMsg, IDS_INVITE_OK, strNick);
        AfxMessageBox(strMsg);
    }
    else
    {
        AfxMessageBox(IDS_INVITE_FAIL);
    }
}

```

```

void CMemberListDlg::OnBtnRenew()
{
    VERIFY(ShowList());
}

void CMemberListDlg::OnClickListMember(NMHDR* pNMHDR, LRESULT* pResult)
{
    DWORD dwPos = ::GetMessagePos();
    CPoint point((int)LOWORD(dwPos), (int)HIWORD(dwPos));
    m_lcMember.ScreenToClient(&point);

    m_nIndex = m_lcMember.HitTest(point);
    if (m_nIndex != -1)
    {
        m_strMessage = m_lcMember.GetItemText(m_nIndex, 0);    //
        SubItem=0
        UpdateData(FALSE);
    }
    *pResult = 0;
}

BOOL CMemberListDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    ASSERT(m_pDoc);

    ASSERT(GetDlgItem(IDC_STATIC_MESSAGE));
    GetDlgItem(IDC_STATIC_MESSAGE)->SetFont(&m_font, FALSE);

    // Prepare List Control for Channels list
    m_lcMember.SetImageList(&m_ilMember, LVSIL_SMALL);

    char* szColumn[] = {"Chat ID", "Information"};
    int nWidth[] = {120, 210};

    LV_COLUMN lvC; // list view column structure
    lvC.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM; //
valid members
    lvC.fmt = LVCFMT_LEFT; // left-align column

    // Add the columns.
    for (int i=0; i < sizeof(nWidth)/sizeof(int); i++)
    {
        lvC.cx = nWidth[i]; // width of column in
pixels
        lvC.iSubItem = i;
        lvC.pszText = szColumn[i];
        if (m_lcMember.InsertColumn(i, &lvC) == -1)
            return NULL;
    }
    VERIFY(ShowList());

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

void CMemberListDlg::OnDeleteitemListMember(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    if (pNMListView->lParam)
    {
        PICS_PROPERTY picsProp = (PICS_PROPERTY)pNMListView->lParam;
        picsProp->Release();
    }
    *pResult = 0;
}

void CMemberListDlg::OnDblclkListMember(NMHDR* pNMHDR, LRESULT* pResult)
{
    DWORD dwPos = ::GetMessagePos();
    CPoint point((int)LOWORD(dwPos), (int)HIWORD(dwPos));
    m_lcMember.ScreenToClient(&point);

    m_nIndex = m_lcMember.HitTest(point);
    if (m_nIndex != -1)
    {
        m_strMessage = m_lcMember.GetItemText(m_nIndex, 0);    //
        SubItem=0
        UpdateData(FALSE);
    }

    CUC2Socket* pCS = m_pDoc->GetSocket();
    if (!pCS)
        return;
    if (pCS->IsQueryOK())
    {
        // I'm alive.
        pCS->SetQueryClient(GetSafeHwnd());
        BeginWaitCursor();
        // m_strMessage = _T("»Ç¿ëÄÚ ¼ÇÁ! ÄĬ$ ÄŦĚ...");
        // UpdateData(FALSE);
        CString strNick = m_lcMember.GetItemText(m_nIndex, 0);
        int len = strNick.GetLength();
        TCHAR* pszNick = strNick.GetBuffer(len);
        if (!pCS->FQueryRealName(pszNick))
        {
            // EndDialog(-1);
        }
    }
    *pResult = 0;
}

void CMemberListDlg::OnDestroy()
{
    CDialog::OnDestroy();
}

// returns the index of the added item
int CMemberListDlg::AddItem(const int i, int iImage, PICS_PROPERTY picsProp,
TCHAR* psz)
{
    if (picsProp)
        picsProp->AddRef();
}

```

```

    LV_ITEM          lvI; // list view item structure
    ::ZeroMemory(&lvI, sizeof(lvI));
    lvI.mask          = LVIF_TEXT | LVIF_IMAGE | LVIF_PARAM | LVIF_STATE;
    lvI.state          = 0;
    lvI.stateMask      = 0;

    lvI.iItem          = i;
    lvI.iSubItem        = 0;
    // The parent window is responsible for storing the text.
    // The list view control will send an LVN_GETDISPINFO
    // when it needs the text to display.
    lvI.pszText         = psz;
    lvI.cchTextMax      = MIC_MAX_USER_NAME_LENGTH; // 63
    lvI.iImage          = iImage;
    lvI.lParam          = (LPARAM)picsProp;

    int iItem = m_lcMember.InsertItem(&lvI);
    if ((iItem == -1) && picsProp) // Insert failure
        picsProp->Release();
    return iItem;
}

BOOL CMemberListDlg::AddMember(PICS_PROPERTY picsProp)
{
    ASSERT(picsProp);
    int i;
    if (FAILED(picsProp->HrGetPrivateData((PVOID*)&i)))
    {
        TRACE0("CMemberListDlg::AddMember - HrGetPrivateData failed.\n");
        return FALSE;
    }
    // Add the channel name..
    CS_PROPDATA cspd;
    if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_MEMBER_NAME)))
        return FALSE;
    CHAR* szName = (CHAR*)cspd.pbData;

    if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_MEMBER_MODE)))
        return FALSE;
    DWORD dwMode = *((DWORD*)cspd.pbData);

    int iImg;
    if (dwMode & CS_MEMBER_HOST)
        iImg = IL_HOST;
    else if (dwMode & CS_MEMBER_SPEAKER)
        iImg = IL_SPEAKER;
    else
        iImg = IL_SPECTATOR;

    int iItem = AddItem(i, iImg, picsProp, szName);
    // m_lcMember.SetItemText(i, 1, szRealName);
    return (iItem != -1);
}

BOOL CMemberListDlg::ShowList()
{
    CUC2Socket* pCS = m_pDoc->GetSocket();

```

```

    if (!pCS)
        return FALSE;
    m_lcMember.DeleteAllItems();

    if (pCS->IsQueryOK())
    {
        // I'm alive.
        pCS->SetQueryClient(GetSafeHwnd());
        BeginWaitCursor();

        if (m_dwChannelID || !m_strChannel.IsEmpty())
        {
            m_strMessage.LoadString(IDS_TRANSFER_MEMBER_CHANNEL);
            UpdateData(FALSE);
            if (m_dwChannelID)
            {
                if (!pCS->FQueryMembersInChannel(m_dwChannelID))
                    EndDialog(-1);
            }
            else
            {
                int nLen = m_strChannel.GetLength();
                if (!pCS->FQueryMembersInChannelName(m_strChannel.GetBuffer(nLen)))
                    EndDialog(-1);
                m_strChannel.ReleaseBuffer();
            }
        }
        else
        {
            m_strMessage.LoadString(IDS_TRANSFER_MEMBER_ALL_CHANNEL);
            UpdateData(FALSE);
            if (!pCS->FQueryListAllUsers())
                EndDialog(-1);
        }
    }
    else
    {
        AfxMessageBox(IDS_QUERY_LINE_BUSY);
        EndDialog(-1);
    }
    // Then UC2Socket will call AddChannel for each item

    return TRUE;
}

void CMemberListDlg::EndOfList()
{
    m_strMessage.LoadString(IDS_DATA_TRANSMISSION_DONE); // "ÄÜ·á Äu¼0
    UpdateData(FALSE);
    EndWaitCursor();
    // ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
}

////////////////////////////////////
// to handle messages from ChatSock

```



```

LRESULT CMemberListDlg::OnQueryMembers(WPARAM wParam, LPARAM lParam)
{
    AddMember((PICS_PROPERTY)lParam);

    m_strCount.Format("%d", m_lcMember.GetItemCount());
    SetDlgItemText(IDC_STATIC_MEMBER_COUNT, m_strCount);
    UpdateData(FALSE);
    return 0;
}

LRESULT CMemberListDlg::OnQueryMembersEnd(WPARAM wParam, LPARAM lParam)
{
    EndOfList();

    /*
    CUC2Socket* pCS = m_pDoc->GetSocket();
    if (!pCS)
        return FALSE;
    if (pCS->IsQueryOK())
    {
        // I'm alive.
        pCS->SetQueryClient(GetSafeHwnd());
        BeginWaitCursor();
        m_strMessage = _T("»Ç¿eÀÚ ¼ÇÁ! ÀÏ.Ş ÁŦÈ....");
        UpdateData(FALSE);
        int nItems = m_lcMember.GetItemCount();
        CString strNick;
        int len;
        m_iRealName = 0;
        for (int i=0; i < nItems; i++)
        {
            strNick = m_lcMember.GetItemText(i, 0);
            len = strNick.GetLength();
            TCHAR* pszNick = strNick.GetBuffer(len);
            if (!pCS->FQueryRealName(pszNick))
            {
                // EndDialog(-1);
            }
        }
    }
    */
    return 0;
}

LRESULT CMemberListDlg::OnQueryGetRealname(WPARAM wParam, LPARAM lParam)
{
    TCHAR* szSource = (TCHAR*)lParam;
    if (!szSource)
        return -1;

    // extern CParser gParser;
    // CString strInfo(szSource);
    // CString strVersion;
    // CString strIP;
    // gParser.CopyBuffer(szSource);
    // gParser.SetLeftToken('`');
    // gParser.GetValueRightToken(strVersion, '@');

```

MemberListDlg.cpp

```
// gParser.GetValueRightToken(strIP, '`');
// strInfo.Format("%s @%s", strVersion, strIP);
// m_lcMember.SetItemText(m_nIndex, 1, strInfo);
// CString strNick = m_lcMember.GetItemText(m_nIndex, 0);
// m_strMessage = strNick + "=" + strInfo;
// UpdateData(FALSE);
// return 0;
}
```

```
LRESULT CMemberListDlg::OnQueryNoMatches(WPARAM wParam, LPARAM lParam)
{
    m_strMessage.LoadString(IDS_QUERY_NO_MATCH);
    SetDlgItemText(IDC_STATIC_MESSAGE, m_strMessage);
    return 0;
}
```

MemberListDlg.h

```
#if
!defined(AFX_MEMBERLISTDLG_H__BCD11B02_C239_11D1_80E2_080009B9F339__INCLUDED_
)
#define AFX_MEMBERLISTDLG_H__BCD11B02_C239_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MemberListDlg.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMemberListDlg dialog

class CUC2Doc;

class CMemberListDlg : public CDialog
{
// Construction
public:
    CMemberListDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{AFX_DATA(CMemberListDlg)
    enum { IDD = IDD_DLG_MEMBER };
    CListCtrl    m_lcMember;
    CString      m_strCount;
    CString      m_strMessage;
    //}}AFX_DATA

    enum
    {
        IL_HOST,
        IL_SPEAKER,
        IL_SPECTATOR,
        IL_COUNT
    } MEMBER_IL_INDEX;

    void SetPPChannel(CUC2Doc* pDoc, const DWORD nID, LPCTSTR
szChannelName=NULL)
    {
        m_pDoc = pDoc;          m_dwChannelID = nID;
        if (szChannelName)
            m_strChannel = szChannelName; }

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CMemberListDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    int          AddItem(const int i, int iImage, PICS_PROPERTY picsProp,
TCHAR* psz);
    BOOL AddMember(PICS_PROPERTY picsProp);
    BOOL ShowList();
```

MemberListDlg.h

```

void EndOfList();

CUC2Doc*      m_pDoc;
DWORD         m_dwChannelID;
CString       m_strChannel;
CFont         m_font;
CImageList    m_ilMember;
// int         m_iRealName;
// int         m_nIndex;

// Generated message map functions
//{{AFX_MSG(CMemberListDlg)
afx_msg void OnBtnInvite();
afx_msg void OnBtnRenew();
afx_msg void OnClickListMember(NMHDR* pNMHDR, LRESULT* pResult);
virtual BOOL OnInitDialog();
afx_msg void OnDeleteitemListMember(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnDblclkListMember(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnDestroy();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
afx_msg LRESULT OnQueryMembers(WPARAM, LPARAM);
afx_msg LRESULT OnQueryMembersEnd(WPARAM, LPARAM);
afx_msg LRESULT OnQueryNoMatches(WPARAM, LPARAM);
afx_msg LRESULT OnQueryGetRealname(WPARAM, LPARAM);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef(AFX_MEMBERLISTDLG_H__BCD11B02_C239_11D1_80E2_080009B9F339__INCLUDED_)

```

Palette.txt

```

{255,255,255,0}, // 0
{8,8,8,0}, // 1
{41,41,41,0}, // 2
{90,90,90,0}, // 3
{165,165,165,0}, // 4
{222,222,222,0}, // 5
{214,206,206,0}, // 6
{156,148,148,0}, // 7
{140,132,132,0}, // 8
{115,107,107,0}, // 9
{90,82,82,0}, // 10
{90,74,74,0}, // 11
{82,66,66,0}, // 12
{33,24,24,0}, // 13
{99,66,66,0}, // 14
{222,123,123,0}, // 15
{214,115,115,0}, // 16
{165,82,82,0}, // 17
{206,99,99,0}, // 18
{57,24,24,0}, // 19
{198,82,82,0}, // 20
{189,74,74,0}, // 21
{173,66,66,0}, // 22
{189,66,66,0}, // 23
{165,49,49,0}, // 24
{198,49,49,0}, // 25
{33,8,8,0}, // 26
{214,49,49,0}, // 27
{198,41,41,0}, // 28
{206,41,41,0}, // 29
{173,33,33,0}, // 30
{181,33,33,0}, // 31
{165,24,24,0}, // 32
{123,16,16,0}, // 33
{189,24,24,0}, // 34
{148,16,16,0}, // 35
{198,16,16,0}, // 36
{140,8,8,0}, // 37
{165,8,8,0}, // 38
{214,8,8,0}, // 39
{8,0,0,0}, // 40
{24,0,0,0}, // 41
{99,0,0,0}, // 42
{115,0,0,0}, // 43
{148,0,0,0}, // 44
{165,0,0,0}, // 45
{198,0,0,0}, // 46
{165,33,24,0}, // 47
{222,140,132,0}, // 48
{231,140,132,0}, // 49
{198,99,90,0}, // 50
{140,41,33,0}, // 51
{189,115,107,0}, // 52
{74,8,0,0}, // 53
{165,115,107,0}, // 54
{132,74,66,0}, // 55
{132,90,82,0}, // 56

```

Palette.txt

```

{82,41,33,0}, // 57
{165,82,66,0}, // 58
{165,90,74,0}, // 59
{132,57,41,0}, // 60
{123,41,24,0}, // 61
{173,140,132,0}, // 62
{140,107,99,0}, // 63
{115,49,33,0}, // 64
{57,24,16,0}, // 65
{132,99,90,0}, // 66
{231,173,156,0}, // 67
{206,148,132,0}, // 68
{206,123,99,0}, // 69
{189,107,82,0}, // 70
{99,41,24,0}, // 71
{115,90,82,0}, // 72
{107,82,74,0}, // 73
{74,49,41,0}, // 74
{222,148,123,0}, // 75
{140,90,74,0}, // 76
{66,41,33,0}, // 77
{132,82,66,0}, // 78
{173,99,74,0}, // 79
{165,90,66,0}, // 80
{99,49,33,0}, // 81
{132,57,33,0}, // 82
{82,33,16,0}, // 83
{74,24,8,0}, // 84
{247,181,156,0}, // 85
{214,148,123,0}, // 86
{206,140,115,0}, // 87
{189,123,99,0}, // 88
{173,107,82,0}, // 89
{57,33,24,0}, // 90
{99,57,41,0}, // 91
{132,66,41,0}, // 92
{57,16,0,0}, // 93
{214,173,156,0}, // 94
{239,181,156,0}, // 95
{173,115,90,0}, // 96
{231,148,115,0}, // 97
{198,115,82,0}, // 98
{132,74,49,0}, // 99
{181,99,66,0}, // 100
{206,189,181,0}, // 101
{173,156,148,0}, // 102
{165,148,140,0}, // 103
{140,123,115,0}, // 104
{132,115,107,0}, // 105
{222,189,173,0}, // 106
{74,57,49,0}, // 107
{206,156,132,0}, // 108
{173,123,99,0}, // 109
{156,99,74,0}, // 110
{189,115,82,0}, // 111
{165,90,57,0}, // 112
{156,82,49,0}, // 113

```

Palette.txt

```

{107,57,33,0}, // 114
{148,74,41,0}, // 115
{189,173,165,0}, // 116
{156,140,132,0}, // 117
{239,206,189,0}, // 118
{165,132,115,0}, // 119
{231,181,156,0}, // 120
{214,165,140,0}, // 121
{132,99,82,0}, // 122
{198,148,123,0}, // 123
{214,148,115,0}, // 124
{107,74,57,0}, // 125
{206,140,107,0}, // 126
{198,132,99,0}, // 127
{99,66,49,0}, // 128
{189,123,90,0}, // 129
{132,82,57,0}, // 130
{206,123,82,0}, // 131
{247,231,222,0}, // 132
{214,198,189,0}, // 133
{181,165,156,0}, // 134
{115,99,90,0}, // 135
{247,189,156,0}, // 136
{239,181,148,0}, // 137
{231,173,140,0}, // 138
{214,156,123,0}, // 139
{222,148,107,0}, // 140
{198,123,82,0}, // 141
{140,82,49,0}, // 142
{239,214,198,0}, // 143
{247,206,181,0}, // 144
{222,181,156,0}, // 145
{239,189,156,0}, // 146
{231,165,123,0}, // 147
{222,156,115,0}, // 148
{255,206,173,0}, // 149
{247,198,165,0}, // 150
{222,173,140,0}, // 151
{239,181,140,0}, // 152
{214,156,115,0}, // 153
{247,214,189,0}, // 154
{239,206,181,0}, // 155
{222,165,123,0}, // 156
{99,66,41,0}, // 157
{239,198,165,0}, // 158
{206,156,115,0}, // 159
{239,181,132,0}, // 160
{247,222,198,0}, // 161
{247,239,231,0}, // 162
{189,181,173,0}, // 163
{231,206,181,0}, // 164
{57,49,41,0}, // 165
{57,41,24,0}, // 166
{107,99,90,0}, // 167
{255,247,231,0}, // 168
{222,214,198,0}, // 169
{255,239,206,0}, // 170

```

Palette.txt

```

{189,156,90,0}, // 171
{214,206,181,0}, // 172
{99,99,82,0}, // 173
{189,198,198,0}, // 174
{222,239,239,0}, // 175
{206,247,255,0}, // 176
{189,231,239,0}, // 177
{214,247,255,0}, // 178
{198,231,239,0}, // 179
{222,247,255,0}, // 180
{189,239,255,0}, // 181
{148,198,214,0}, // 182
{198,222,231,0}, // 183
{165,206,222,0}, // 184
{66,107,123,0}, // 185
{33,57,66,0}, // 186
{181,222,239,0}, // 187
{82,123,140,0}, // 188
{115,173,198,0}, // 189
{49,90,107,0}, // 190
{181,231,255,0}, // 191
{165,222,247,0}, // 192
{148,198,222,0}, // 193
{82,132,156,0}, // 194
{24,41,49,0}, // 195
{33,66,82,0}, // 196
{156,189,206,0}, // 197
{173,222,247,0}, // 198
{165,214,239,0}, // 199
{140,189,214,0}, // 200
{132,181,206,0}, // 201
{123,173,198,0}, // 202
{74,123,148,0}, // 203
{99,165,198,0}, // 204
{49,82,99,0}, // 205
{66,115,140,0}, // 206
{123,165,189,0}, // 207
{41,57,66,0}, // 208
{214,239,255,0}, // 209
{132,173,198,0}, // 210
{107,148,173,0}, // 211
{99,140,165,0}, // 212
{74,115,140,0}, // 213
{41,66,82,0}, // 214
{33,74,99,0}, // 215
{156,181,198,0}, // 216
{198,231,255,0}, // 217
{99,148,181,0}, // 218
{24,49,66,0}, // 219
{123,156,181,0}, // 220
{57,90,115,0}, // 221
{115,156,189,0}, // 222
{148,165,181,0}, // 223
{24,33,41,0}, // 224
{132,140,148,0}, // 225
{107,123,140,0}, // 226
{99,115,132,0}, // 227

```


Palette.txt

```
{8,16,24,0}, // 228
{90,99,115,0}, // 229
{206,99,115,0}, // 230
{214,66,90,0}, // 231
{206,82,99,0}, // 232
{198,74,90,0}, // 233
{222,41,66,0}, // 234
{214,123,132,0}, // 235
{214,57,74,0}, // 236
{181,41,57,0}, // 237
{206,41,57,0}, // 238
{181,24,41,0}, // 239
{165,8,24,0}, // 240
{206,99,107,0}, // 241
{214,90,99,0}, // 242
{173,57,66,0}, // 243
{181,57,66,0}, // 244
{198,82,90,0}, // 245
{206,82,90,0}, // 246
{198,74,82,0}, // 247
{181,49,57,0}, // 248
{198,49,57,0}, // 249
{181,41,49,0}, // 250
{181,33,41,0}, // 251
{214,33,41,0}, // 252
{214,16,24,0}, // 253
{206,8,16,0}, // 254
{0,0,0,0}, // 255
```

PaletteDlg.cpp

```
// PaletteDlg.cpp : implementation file
//

#include "stdafx.h"
#include "BMC.h"
#include "PaletteDlg.h"

#include "BMCView.h"
#include "../UC2Ani/DIBPal.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CPaletteDlg dialog

CPaletteDlg::CPaletteDlg(CWnd* pParent /*=NULL*/)
: CDialog(CPaletteDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPaletteDlg)
    m_sbBlue = 0;
    m_sbGreen = 0;
    m_stRGB = _T("");
    m_sbRed = 0;
    m_bShowIndex = FALSE;
    //}}AFX_DATA_INIT
}

void CPaletteDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPaletteDlg)
    DDX_Scroll(pDX, IDC_SB_PAL_BLUE, m_sbBlue);
    DDX_Scroll(pDX, IDC_SB_PAL_GREEN, m_sbGreen);
    DDX_Text(pDX, IDC_ST_PAL_RGB, m_stRGB);
    DDX_Scroll(pDX, IDC_SB_PAL_RED, m_sbRed);
    DDX_Check(pDX, IDC_CHECK_SHOW_INDEX, m_bShowIndex);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPaletteDlg, CDialog)
    //{{AFX_MSG_MAP(CPaletteDlg)
    ON_WM_PAINT()
    ON_BN_CLICKED(IDC_CHECK_SHOW_INDEX, OnCheckShowIndex)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// CPaletteDlg message handlers

BOOL CPaletteDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
}
```

```

GetClientRect(&m_rcPal);

CWnd* pW = GetDlgItem(IDC_SB_PAL_RED);
ASSERT(pW);
CRect rcSB;
pW->GetClientRect(&rcSB);
m_rcPal.right -= (rcSB.Width() + 10);

return TRUE; // return TRUE unless you set the focus to a control
             // EXCEPTION: OCX Property Pages should return FALSE
}

void CPaletteDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    if (!m_pPal)
        return;

    m_pPal->Draw(&dc, m_rcPal, FALSE, m_bShowIndex);

    // Do not call CDialog::OnPaint() for painting messages
}

void CPaletteDlg::OnCheckShowIndex()
{
    CButton* pB = (CButton*)GetDlgItem(IDC_CHECK_SHOW_INDEX);
    ASSERT(pB);
    m_bShowIndex = (pB->GetCheck() == 1);
    InvalidateRect(&m_rcPal, FALSE);
}

```

PaletteDlg.h

```

#if
!defined(AFX_PALETTEDLG_H__460C5624_91A5_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_PALETTEDLG_H__460C5624_91A5_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PaletteDlg.h : header file
//

class CDIBPal;

////////////////////////////////////
// CPaletteDlg dialog

class CPaletteDlg : public CDialog
{
// Construction
public:
    CPaletteDlg(CWnd* pParent = NULL);    // standard constructor

public:
    void SetPalette(CDIBPal* pPal)      { m_pPal = pPal; }

// Dialog Data
    //{AFX_DATA(CPaletteDlg)
    enum { IDD = IDD_PALETTE };
    int         m_sbBlue;
    int         m_sbGreen;
    CString     m_stRGB;
    int         m_sbRed;
    BOOL        m_bShowIndex;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CPaletteDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    CDIBPal*     m_pPal;
    CRect        m_rcPal;

    // Generated message map functions
    //{AFX_MSG(CPaletteDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg void OnCheckShowIndex();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}

```

PaletteDlg.h

// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //

!defined(AFX_PALETTEDLG_H__460C5624_91A5_11D1_80E2_080009B9F339__INCLUDED_)

```

// Parser.cpp
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//UNICHAT INC
//=====

//ifndef _SDK
#include "stdafx.h"
//endif
#include "Parser.h"
#include <lzexpand.h>

const int MAX_EXPANDED_FILESIZE = 120*1024;

CTextFileBuffer::CTextFileBuffer(const int nLineBufferLen)
{
    m_nLineBufferLen = nLineBufferLen; // Maximum String Length
    m_pLineBuffer    = m_nLineBufferLen ? new char[m_nLineBufferLen]
: NULL;
    m_lFileSize       = 0L;
    m_pFileBuffer     = NULL;
    m_pCursor         = NULL;
}

CTextFileBuffer::~CTextFileBuffer()
{
    if (m_pLineBuffer)
        delete [] m_pLineBuffer;
    if (m_pFileBuffer)
        delete [] m_pFileBuffer;
}

BOOL CTextFileBuffer::Load(LPCSTR szFile)
{
    TRY
    {
        CFile f;
        if (!f.Open(szFile, CFile::modeRead | CFile::shareDenyWrite))
            // | CFile::typeText))
            {
                CString strMsg;
                strMsg.Format("File Open Failure: %s", szFile);
                AfxMessageBox(strMsg);
                return FALSE;
            }
        WORD wFileType;
        int iBytes = f.Read(&wFileType, sizeof(wFileType));
        if (iBytes != sizeof(wFileType))
            {
                CString strMsg;
                strMsg.Format("File read failure! %s: %d bytes read",
szFile, iBytes);
                AfxMessageBox(strMsg);
                return FALSE;
            }
    }
}

```

```

        m_lFileSize = f.GetLength(); // points to the end of file
        TRACE("FileSize=%ld bytes\n", m_lFileSize);

        BOOL bLZ = (wFileType == 0x5a53); // 'SZ' LZ Compressed
        UINT hLZFile;
        if (bLZ)
        {
            f.Close();
            OFSTRUCT ofStructSrc;
            hLZFile = ::LZOpenFile((char*)szFile, &ofStructSrc,
OF_READ);
        }
        if (bLZ)
            ::LZSeek(hLZFile, 0, 0); // seek to beginning of file
        else
            f.Seek(0, CFile::begin);
        if (bLZ)
        {
            // Set max. buffer 120KB since it is a text file...
            m_lFileSize = MAX_EXPANDED_FILESIZE;
            m_pFileBuffer = new char[m_lFileSize];
            iBytes = ::LZRead(hLZFile, m_pFileBuffer, m_lFileSize);
            ASSERT(iBytes < m_lFileSize);
            m_lFileSize = iBytes;
        }
        else
        {
            m_pFileBuffer = new char[m_lFileSize+1];
            iBytes = f.Read(m_pFileBuffer, m_lFileSize);
            if (iBytes != (int)m_lFileSize)
            {
                CString strMsg;
                strMsg.Format("File read failure! %s: %d bytes read",
szFile, iBytes);
                AfxMessageBox(strMsg);
                return FALSE;
            }
        }
        m_pFileBuffer[m_lFileSize] = NULL;
        if (bLZ)
        {
            ::LZClose(hLZFile); // Commenting out this line
prevented error in NT system.
        }
        else
        {
            f.Close();
        }
        SeekToBegin(); // m_pCursor = m_pFileBuffer;
        return TRUE; // Omitting this line was the bug! only for
Release version.
    }
    CATCH( CFileException, e )
    {
        #ifdef _DEBUG

```

```

        afxDump << "File could not be opened " << e->m_cause <<
"\n";
        #endif
        CString strMsg;
        strMsg.Format("File read failure! %s", szFile);
        AfxMessageBox(strMsg);
        return FALSE;
    }
END_CATCH
}

char* CTextFileBuffer::ReadString()
{
    if (!m_pLineBuffer)
        return NULL;
    char* pNL = strstr(m_pCursor, "\r\n");
    if (!pNL)
        return NULL;
    int nSize = pNL - m_pCursor + 1;
    if (nSize > m_nLineBufferLen)
        nSize = m_nLineBufferLen;
    ::CopyMemory(m_pLineBuffer, m_pCursor, nSize);
    m_pLineBuffer[nSize-1] = NULL;
    m_pCursor += (nSize + 1);
    return m_pLineBuffer;
}

////////////////////////////////////
/////
CParser::CParser(const int mchars)
{
    m_bMsgOn    = TRUE;
    m_raw       = 0;
    m_MaxChars  = mchars;
    m_szBuf     = new char[mchars];
}

// #ifndef _SDK
// returns TRUE if the file exists
BOOL CParser::ExistFile(const char* szFile)
{
    CFileStatus status;
    BOOL bFound = CFile::GetStatus(szFile, status);
    if (!bFound)
    {
        TRACE1("File(%s) not found!\n", szFile);
        if (m_bMsgOn)
        {
            CString msg;
            msg.Format("File(%s) not found!", szFile);
            AfxMessageBox(msg);
        }
    }
    return bFound;
}

// #endif // _SDK

```



```

// Initialization function to be called first
void CParser::CopyBuffer(const char* line)
{
    char* p = (char*)line;
    while (IsWhiteSpace(*p)) // remove heading spaces
        p++;
    m_raw = m_szBuf; // set m_raw string to be parsed
    int len = strlen(p);
    if (len > m_MaxChars)
    {
        len = m_MaxChars;
        ::CopyMemory(m_szBuf, p, len);
        m_szBuf[len-1] = NULL;
    }
    else
        strcpy(m_szBuf, p);
}

char* CParser::SetLeftToken(const char lefttok)
{
    char* p = strchr(m_raw, lefttok); // Although lefttok was not
    found, // I don't want
    to lose m_raw.
    if (p)
    {
        p++; // advance pointer to the
        next char
        while (IsWhiteSpace(*p)) // remove heading spaces for next
        parsing
            p++;
        m_raw = p; // save current position
    }
    return p; // If lefttok was not found, return NULL
}

char* CParser::SetLeftToken(char* lefttokstr)
{
    char* p = strstr(m_raw, lefttokstr);
    if (p)
    {
        p += strlen(lefttokstr);
        while (IsWhiteSpace(*p)) // remove heading spaces
            p++;
        m_raw = p;
    }
    return p; // If lefttok was not found, return NULL
}

char* CParser::FindToken(const char rtok1, const char rtok2)
{
    char* p = strchr(m_raw, rtok1);
    if (rtok2)
    {
        char* q = strchr(m_raw, rtok2);
        // If rtok1 not found or found all two tokens then use the left
one

```

```

        // In case, ")()()..."
        if (!p || (q && p && p > q))
            p = q;
    }
    m_lastToken = p ? *p : NULL;
    return p;
}

void CParser::RemoveSpaces(char** pp)
{
    char* p = *pp;
    char* q = p;          // remove trailing spaces
    q--;
    while (IsWhiteSpace(*q))
        *q-- = NULL;
    *p++ = NULL;          // Nullify the token char and advance to the
next char
    while (IsWhiteSpace(*p)) // remove heading spaces
        p++;
    *pp = p;
}

// some bytes of buf will be overwritten!
// Get value before right token: character version
char* CParser::GetValueRightToken(char& cVal, const char rtok1, const char
rtok2)
{
    char* p = FindToken(rtok1, rtok2);
    if (p)          // found the 'right' token
    {
        RemoveSpaces(&p);
        cVal = *m_raw;
        m_raw = p;          // save current position
    }
    else
        cVal = *m_raw;      // for null-terminating string...
    // Although righttok not found, try to
convert...
    return p;    // if righttok was not found, return NULL
}

// integer version
char* CParser::GetValueRightToken(int& iVal, const char rtok1, const char
rtok2)
{
    char* p = FindToken(rtok1, rtok2);
    if (p)          // found the 'right' token
    {
        RemoveSpaces(&p);
        iVal = atoi(m_raw);
        m_raw = p;          // save current position
    }
    else
        iVal = atoi(m_raw); // Although righttok not found, try to
convert...
    return p;    // if righttok was not found, return NULL
}

```

```

// word version
char* CParser::GetValueRightToken(WORD& wVal, const char rtok1, const char
rtok2)
{
    char* p = FindToken(rtok1, rtok2);
    if (p)
        // found the 'right' token
        {
            RemoveSpaces(&p);
            wVal = (WORD)atoi(m_raw);
            m_raw = p;
            // save current position
        }
    else
        wVal = (WORD)atoi(m_raw);
    // Although righttok not found, try
to convert...
    return p;
    // if righttok was not found, return NULL
}

// long version
char* CParser::GetValueRightToken(long& lVal, const char rtok1, const char
rtok2)
{
    char* p = FindToken(rtok1, rtok2);
    if (p)
        {
            RemoveSpaces(&p);
            lVal = atol(m_raw);
            m_raw = p;
        }
    else
        lVal = atol(m_raw);
    // Although righttok not found, try to
convert...
    return p;
}

// double version
char* CParser::GetValueRightToken(double& fVal, const char rtok1, const char
rtok2)
{
    char* p = FindToken(rtok1, rtok2);
    if (p)
        {
            RemoveSpaces(&p);
            fVal = atof(m_raw);
            m_raw = p;
        }
    else
        fVal = atof(m_raw);
    // Although righttok not found, try to
convert...
    return p;
}

// char string version
// The parameter szVal should have enough spaces to get the characters from
buf
// ABCD , 123,
// <-- ^p --> remove white spaces surrounding token

```

```

//      ^q      ^m_raw
char* CParser::GetValueRightToken(char* szVal, const char rtok1, const char
rtok2)
{
    char* p = FindToken(rtok1, rtok2);
    if (p)
    {
        RemoveSpaces(&p);
        lstrcpy(szVal, m_raw);
        m_raw = p;
    }
    else
        lstrcpy(szVal, m_raw); // Although righttok not found, try to
convert...
    return p;
}

////////////////////////////////////
////////////////////////////////////
// MFC Specific
// #ifndef _SDK
char* CParser::GetValueRightToken(CString& strVal, const char rtok1, const
char rtok2)
{
    char* p = FindToken(rtok1, rtok2);
    if (p)
    {
        RemoveSpaces(&p);
        strVal = m_raw; // CString will prepare space...
        m_raw = p;
    }
    else
        strVal = m_raw; // Although righttok not found, try to
convert...
    return p;
}

char* CParser::GetValueRightToken(CPoint& ptVal, const char rtok1, const char
rtok2)
{
    if (GetStruct(ptVal) != 0)
        return NULL;
    char c;
    return GetValueRightToken(c, rtok1, rtok2); // to remove rtok?
}

char* CParser::GetValueRightToken(CSize& ptVal, const char rtok1, const char
rtok2)
{
    if (GetStruct(ptVal) != 0)
        return NULL;
    char c;
    return GetValueRightToken(c, rtok1, rtok2); // to remove rtok?
}

char* CParser::GetValueRightToken(CRect& ptVal, const char rtok1, const char
rtok2)

```

```

{
    if (GetStruct(ptVal) != 0)
        return NULL;
    char c;
    return GetValueRightToken(c, rtok1, rtok2);    // to remove rtok?
}

// Be sure NOT to call SetLeftToken before calling this function!
int CParser::GetStruct(CPoint& ptVal)
{
    if (!SetLeftToken('(')) return -1;
    if (!GetValueRightToken(ptVal.x, ',')) return -1;
    if (!GetValueRightToken(ptVal.y, ')')) return -1;
    return 0;
}

int CParser::GetStruct(CSize& sVal)
{
    if (!SetLeftToken('(')) return -1;
    if (!GetValueRightToken(sVal.cx, ',')) return -1;
    if (!GetValueRightToken(sVal.cy, ')')) return -1;
    return 0;
}

// Be sure not to call SetLeftToken before calling this function!
int CParser::GetStruct(CRect& rcVal)
{
    if (!SetLeftToken('(')) return -1;
    if (!GetValueRightToken(rcVal.left, ',')) return -1;
    if (!GetValueRightToken(rcVal.top, ',')) return -1;
    if (!GetValueRightToken(rcVal.right, ',')) return -1;
    if (!GetValueRightToken(rcVal.bottom, ')')) return -1;
    return 0;
}

// #endif // _SDK

// Count occurrences of a specified char in m_raw string up to 'upto' char
// Ex.) count = CountOccurrencesUpto(')', ';')
int CParser::CountOccurrencesUpto(const char ch, const char upto)
{
    int count = 0;
    char* p = m_raw;

    while (*p && (*p != upto))
        if (*p++ == ch)
            count++;
    return count;
}

// Default Implementation
// comment line is a string that begins with ';' or "/*"
BOOL CParser::IsCommentLine() const
{
    char* p = m_szBuf;
    while (IsWhiteSpace(*p)) // skip heading spaces
        p++;
    return(*p != 0 || *p == ';' || (*p == '/' && *(p+1) == '/'));
}

```

}

[illegible]

Parser.h

```
//
//      CParser: General Line Parser
//=====
//      (C) Programmed by Kim,
//      UNICHAT Lab
//      Information Technology Institute
//      UNICHAT
//=====
//
#ifdef __PARSER_H
#define __PARSER_H

//ifndef _SDK
#include "stdafx.h"
//endif
#include <windows.h>

////////////////////////////////////
// Read whole texts in the file into the memory
class CTextFileBuffer : public COject
{
public:
    CTextFileBuffer(const int nLineBufferLen=0);
    virtual ~CTextFileBuffer();

    BOOL Load(LPCSTR szFile);
    char* GetString() { return m_pLineBuffer; }
    char* ReadString();
    char* GetWholeText() { return m_pFileBuffer; }
    void SeekToBegin() { m_pCursor = m_pFileBuffer; }

protected:
    int m_nLineBufferLen; // Maximum String Length
    char* m_pLineBuffer; // Current text line
    long m_lFileSize;
    char* m_pFileBuffer;
    char* m_pCursor;
};

////////////////////////////////////
// Line by line parser
class CParser
{
public:
    CParser(const int mchars = 1024);
    ~CParser() { delete [] m_szBuf; }

    int GetMaxBuffer() const { return m_MaxChars; }
    void CopyBuffer(const char* line);
    char GetLastToken() { return m_lastToken; }
    void SetMessageBoxOption(BOOL bMsgOn=TRUE) { m_bMsgOn = bMsgOn; }
    BOOL IsMsgOn() const { return m_bMsgOn; }
//ifndef _SDK
    BOOL ExistFile(const char* szFile);
//endif
    char* GetCurrent() { return m_raw; }
    void SetCurrent(char* p) { m_raw = p; }
```

```

char* FindToken(const char rtok1, const char rtok2=NULL);
char* FindChar(const char tok);
char* SetLeftToken(const char lefttok);
char* SetLeftToken(char* lefttokstr);
// You can assign primary token and secondary token
char* GetValueRightToken(char& cVal,      const char rtok1, const char
rtok2 = 0);
char* GetValueRightToken(int& iVal, const char rtok1, const char rtok2
= 0);
char* GetValueRightToken(WORD& wVal,      const char rtok1, const char
rtok2 = 0);
char* GetValueRightToken(long& lVal,      const char rtok1, const char
rtok2 = 0);
char* GetValueRightToken(double& fVal, const char rtok1, const char
rtok2 = 0);
char* GetValueRightToken(char* szVal,      const char rtok1, const char
rtok2 = 0);
#ifdef _SDK
char* GetValueRightToken(CString& strVal, const char rtok1, const char
rtok2 = 0);
char* GetValueRightToken(CPoint& ptVal, const char rtok1, const char
rtok2 = 0);
char* GetValueRightToken(CSize& ptVal, const char rtok1, const char
rtok2 = 0);
char* GetValueRightToken(CRect& ptVal, const char rtok1, const char
rtok2 = 0);
int      GetStruct(CPoint& ptVal);
int      GetStruct(CSize& sVal);
int      GetStruct(CRect& rcVal);
#endif
int      CountOccurrencesUpto(const char ch, const char upto);
virtual BOOL IsCommentLine() const;
BOOL IsWhiteSpace(const char c) const { return (c == ' ' || c ==
'\t'); }

protected:
void RemoveSpaces(char** pp);
BOOL m_bMsgOn; // reports error messages through MessageBoxes
char m_lastToken;
private:
char* m_raw; // raw points to the raw string to be parsed
char* m_szBuf; // Buffer for one line
int m_MaxChars; // maximum chars per line
};

#endif // __PARSER_H

/* ### Usage #####

CParser parser(256);
...

BOOL CTestDoc::OnNewDocument()
{
    ...
    TRY
    {

```


Parser.h

```

CStdioFile f("sample.txt", CFile::modeRead | CFile::typeText);
char* szVal = new char[parser.GetMaxBuffer()];
int      iVal;
double   fVal;
CPoint   ptVal;
while(f.ReadString(szBuf, parser.GetMaxBuffer()))
{
    parser.CopyBuffer(szBuf);          // "°Å°İ=1,3.4,(12,345);"
    if (parser.IsCommentLine())
        continue;

    parser.GetValueRightToken(szVal, '=');
    parser.SetLeftToken('=');
    parser.GetValueRightToken(iVal, ',');
    parser.GetValueRightToken(fVal, ',');
    parser.GetValueRightToken(ptVal);
}
return TRUE;
f.Close();
delete [] szBuf;
}
CATCH( CFileException, e )
{
    #ifdef _DEBUG
        afxDump << "File could not be opened " << e->m_cause <<
"\n";
    #endif
    delete [] szBuf;
    return FALSE;
}
END_CATCH
}
*/

```

```

// PPActor.cpp : implementation file
//
//=====
//      (C) Programmed by Kim, Soomin, Mar 1998
//      Information Technology Institute
//      UNICHAT
//=====

#include "stdafx.h"
#include "resource.h"
#include "PPActor.h"
#include "PSJoinChannel.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/PhSprite.h"
#include "ResMan.h"
#include "Behavior.h"    // CActorDesc

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

IMPLEMENT_DYNCREATE(CPPActor, CPropertyPage)

////////////////////////////////////
// CPPActor property page

CPPActor::CPPActor() : CPropertyPage(CPPActor::IDD)
{
    TRACE0("CPPActor::CPPActor()\n");
    //{AFX_DATA_INIT(CPPActor)
    m_nAge = 20;
    //}AFX_DATA_INIT

    /*
        m_strName = _T("");
        m_strProfile = _T("");
        m_strSex = _T("");
        m_strHomePage = _T("");
        m_strChatID = _T("");
        m_strUnitelID = _T("");
        m_strVersion = _T("");
    */

    m_nCharID = 0;
    m_nCellID = 0;
    m_nSex = 0;
    m_pPSActor = NULL;
}

CPPActor::~CPPActor()
{
    TRACE0("CPPActor::~CPPActor()\n");
    if (m_pPSActor)
        delete m_pPSActor;
}

```

```

void CPPActor::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPActor)
    DDX_Control(pDX, IDC_COMBO_SEX, m_cbSex);
    DDX_Control(pDX, IDC_LB_ACTORS, m_lbActors);
    DDX_Control(pDX, IDC_SB_ACTOR, m_sbActor);
    DDX_Text(pDX, IDC_EDIT_AGE, m_nAge);
    DDV_MinMaxUInt(pDX, m_nAge, 0, 200);
    DDX_Text(pDX, IDC_EDIT_NAME, m_strName);
    DDV_MaxChars(pDX, m_strName, 20);
    DDX_Text(pDX, IDC_EDIT_PROFILE, m_strProfile);
    DDV_MaxChars(pDX, m_strProfile, 256);
    DDX_Text(pDX, IDC_EDIT_HOMEPAGE, m_strHomePage);
    DDX_Text(pDX, IDC_EDIT_CHAT_ID, m_strChatID);
    DDV_MaxChars(pDX, m_strChatID, 20);
    DDX_Text(pDX, IDC_ST_VERSION, m_strVersion);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPActor, CPropertyPage)
    //{{AFX_MSG_MAP(CPPActor)
    ON_WM_ERASEBKGD()
    ON_WM_SIZE()
    ON_EN_CHANGE(IDC_EDIT_AGE, OnChangeEditAge)
    ON_EN_CHANGE(IDC_EDIT_NAME, OnChangeEditName)
    ON_EN_CHANGE(IDC_EDIT_PROFILE, OnChangeEditProfile)
    ON_LBN_SELCHANGE(IDC_LB_ACTORS, OnSelchangeLbActors)
    ON_WM_PAINT()
    ON_WM_HSCROLL()
    ON_CBN_SELCHANGE(IDC_COMBO_SEX, OnSelchangeComboSex)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CPPActor::OnEraseBkgnd(CDC* pDC)
{
    CPropertyPage::OnEraseBkgnd(pDC);

    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);
    return pPSJC->OnPageEraseBkgnd(pDC);
}

BOOL CPPActor::OnInitDialog()
{
    TRACE0("CPPActor::OnInitDialog()\n");
    CPropertyPage::OnInitDialog();

    m_cbSex.SetCurSel(m_nSex);

    for (int i=0; i < gResMan.GetNumActorDescs(); i++)
    {
        CActorDesc* pAD = gResMan.GetActorDesc(i);
        if (pAD)
        {
            CString* pNick = pAD->GetNick();
            if (pNick)

```

```

        m_lbActors.AddString(*pNick);
    }
    m_lbActors.SetCurSel(0);

    m_rcActor.left = 10;
    m_rcActor.top = 10;
    UpdateActorImage();

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

/*
HBRUSH CPPActor::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);

    if (nCtlColor == CTLCOLOR_STATIC)
    {
        pDC->SetBkMode(TRANSPARENT);
        return (HBRUSH)*pPSJC->GetNullBrush();
    }
    else
    {
        HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
        return hbr;
    }
}
*/

void CPPActor::OnSize(UINT nType, int cx, int cy)
{
    TRACE("CPPActor::OnSize(%d,%d,%d)\n", nType, cx, cy);
    CPropertyPage::OnSize(nType, cx, cy);

    /*
    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);
    if (pPSJC->GetDIBBack())
    {
        SetWindowPos(NULL, 0, 0, pPSJC->GetDIBBack()->GetWidth(), pPSJC-
>GetDIBBack()->GetHeight(),
                        SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
    }
    */
}

void CPPActor::OnChangeEditAge()
{
    SetModified();
}

void CPPActor::OnChangeEditName()
{
    SetModified();
}

```

```

void CPPActor::OnChangeEditProfile()
{
    SetModified();
}

void CPPActor::OnSelchangeLbActors()
{
    m_nCharID = m_lbActors.GetCurSel();
    m_nCellID = 0;
    UpdateActorImage();
    SetModified();
}

void CPPActor::OnSelchangeComboSex()
{
    m_nSex = m_cbSex.GetCurSel();
    TRACE("Sex:%d\n", m_nSex);
    SetModified();
}

void CPPActor::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_ACTOR);

    if (pScrollBar == pSB)
    {
        if (!m_pPSActor)
            return;
        int nCells = m_pPSActor->GetNumCells();
        int nMin, nMax;
        pSB->GetScrollRange(&nMin, &nMax);
        switch (nSBCode)
        {
            case SB_THUMBPOSITION:
                m_nCellID = nPos;
                pSB->SetScrollPos(nPos);
                break;
            case SB_LINELEFT:
                m_nCellID = pSB->GetScrollPos() - 1;
                if (m_nCellID < nMin)
                    m_nCellID = nMax; // Wrap
                pSB->SetScrollPos(m_nCellID);
                break;
            case SB_LINERIGHT:
                m_nCellID = pSB->GetScrollPos() + 1;
                if (m_nCellID > nMax)
                    m_nCellID = nMin; // Wrap
                pSB->SetScrollPos(m_nCellID);
                break;
            case SB_PAGELEFT:
                m_nCellID = pSB->GetScrollPos() - (nMax - nMin) / 5;
                if (m_nCellID < nMin)
                    m_nCellID = nMin;
                pSB->SetScrollPos(m_nCellID);
                break;
            case SB_PAGERIGHT:

```

```

        m_nCellID = pSB->GetScrollPos() + (nMax - nMin) / 5;
        if (m_nCellID > nMax)
            m_nCellID = nMax;
        pSB->SetScrollPos(m_nCellID);
        break;
    }
    InvalidateRect(&m_rcActor, FALSE);
    UpdateData(FALSE);
}
CPropertyPage::OnHScroll(nSBCode, nPos, pScrollBar);
}

// m_nCharID can be set externally by calling CPSJoinChannel::SetMemberInfo
void CPPActor::UpdateActorImage()
{
    ActorDesc* pAD = gResMan.GetActorDesc(m_nCharID);
    if (!pAD)
    {
        TRACE0("CPPActor - ActorDesc not found!\n");
        return;
    }
    CString* pstrRes = pAD->GetResName();
    if (!pstrRes)
    {
        TRACE0("CPPActor - ActorDesc - ResName not found!\n");
        return;
    }
    CString strResName(*pstrRes);
    // We'll not reuse DIB here:
    // Since this image is not necessary to be resident in memory.
    CPhasedSprite* pPS = gResMan.LoadPhasedSprite(strResName, FALSE);
    if (!pPS)
    {
        delete pPS;
        strResName += " not found!";
        AfxMessageBox(strResName);
        return;
    }

    if (m_pPSActor)
        delete m_pPSActor; // Delete previously allocated resource
    m_pPSActor = pPS;
    m_nCellID = 0;
    m_pPSActor->SetCell(m_nCellID);
    CRect rc;
    m_pPSActor->GetRect(rc);
    m_rcActor.right = m_rcActor.left + rc.Width();
    m_rcActor.bottom = m_rcActor.top + rc.Height();

    CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_ACTOR);
    ASSERT(pSB);
    pSB->SetWindowPos(NULL, m_rcActor.left, m_rcActor.bottom+2,
                      m_rcActor.Width()*2, 15,
                      SWP_NOZORDER | SWP_NOACTIVATE);
    pSB->SetScrollRange(0, m_pPSActor->GetNumCells()-1);
    pSB->SetScrollPos(0);
}

```

```

        m_lbActors.SetCurSel(m_nCharID);

        InvalidateRect(&m_rcActor);
    }

void CPPActor::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);
    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (pPSJC->GetPalette())
    {
        pPalOld = dc.SelectPalette(pPSJC->GetPalette(), FALSE); //
bForceBackground = FALSE
        // pDC->RealizePalette(); // we realize in response to
WM_QUERYNEWPALETTE
    }

    if (m_pPSActor)
    {
        m_pPSActor->SetCell(m_nCellID);
        CPoint ptLT(m_rcActor.left, m_rcActor.top);
        m_pPSActor->Draw(&dc, ptLT);
    }

    if (pPalOld)
        dc.SelectPalette(pPalOld, FALSE);
    // Do not call CPropertyPage::OnPaint() for painting messages
}

BOOL CPPActor::OnApply()
{
    // Satya Changes Start:
    // At any moment if click ok button on the Actor Acreen it
    // should activate the next TAB i.e Room List..

    CPSJoinChannel *pParent = (CPSJoinChannel*)this->GetParent();
    ASSERT(pParent);

    if (pParent->SetActivePage(1))
    {
        return TRUE;
    }
    else
    {
        return CPropertyPage::OnApply();
    }
}

```

Ppactor.h

```
// PPActor.h : header file
//
//=====
//      (C) Programmed by Kim, Soomin, Mar 1998
//      Information Technology Institute
//      UNICHAT.COM
//=====

#ifndef __PPACTOR_H__
#define __PPACTOR_H__

////////////////////////////////////
// CPPActor dialog

class CPhasedSprite;

class CPPActor : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPActor)

// Construction
public:
    CPPActor();
    ~CPPActor();

    int          m_nCharID;
    int          m_nSex;           // 0: Male, 1: Female

// Dialog Data
   //{{AFX_DATA(CPPActor)
    enum { IDD = IDD_PP_ACTOR };
    CComboBox    m_cbSex;
    CListBox     m_lbActors;
    CScrollBar   m_sbActor;
    UINT         m_nAge;
    CString      m_strName;
    CString      m_strProfile;
    CString      m_strHomePage;
    CString      m_strChatID;
    CString      m_strUnitelID;
    CString      m_strVersion;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPPActor)
    public:
    virtual BOOL OnApply();
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    void          UpdateActorImage();

    int          m_nCellID;

```


Ppactor.h

```

CPhasedSprite*      m_pPSActor;
CRect               m_rcActor;

// Generated message map functions
//{{AFX_MSG(CPPActor)
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
virtual BOOL OnInitDialog();
afx_msg void OnSize(UINT nType, int cx, int cy);
afx_msg void OnChangeEditAge();
afx_msg void OnChangeEditName();
afx_msg void OnChangeEditProfile();
afx_msg void OnSelchangeLbActors();
afx_msg void OnPaint();
afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar);
afx_msg void OnSelchangeComboSex();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

#endif // __PPACTOR_H__

```

```

// PPChannel.cpp : implementation file
//
//=====
//      (C) Programmed by Kim, Soomin, Mar 1998
//      Information Technology Institute
//      UNICHAT.COM
//=====

#include "stdafx.h"
#include "resource.h"
#include "PPChannel.h"
#include "PSJoinChannel.h"
#include "UC2CS.h"
#include "MemberListDlg.h"
#include "InputPassword.h"
#include "UC2.h" // RegGetLastStageID()
#include "ResMan.h"
#include "Parser.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;
// SATYA CHANGES START
extern CParser gParser;
// SATYA CHANGES END

IMPLEMENT_DYNCREATE(CPPChannel, CPropertyPage)

////////////////////////////////////
// CPPChannel property page

CPPChannel::CPPChannel() : CPropertyPage(CPPChannel::IDD)
{
    TRACE0("CPPChannel::CPPChannel()\n");
    //{AFX_DATA_INIT(CPPChannel)
    m_strCount = _T("");
    m_strMessage = _T("");
    m_strStageID = _T("");
    //}}AFX_DATA_INIT
    m_font.CreateFont(-13, 0, 0, 0, FW_BOLD,
        FALSE, FALSE, 0, // bItalic, bUnderline, cStrikeOut
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH || FF_DONTCARE,
#ifdef _KOREAN
        "±¼.²Ä¼");
#else
        "Times New Roman");
#endif
    m_ilChannel.Create(IDB_IL_CHANNEL, 16, 1, CLR_NONE);
    m_nIndex = -1;
    m_nUsers = 0;
    m_bMUD = FALSE;
    // SATYA CHANGES START

```

```

// create the Images for Tree Control :
m_ilChnCatTree.Create(IDB_IL_CREATECHANNEL, 16, 1, CLR_NONE);
// set the default values for Room Category

m_ChnCatType.LANG=0;
m_ChnCatType.CROOT=0;
m_ChnCatType.CCHILD=0;
// Re set the Room Category tree
ReSetChannelCat();
// SATYA CHANGES END

}

CPPChannel::~CPPChannel()
{
    TRACE0("CPPChannel::~CPPChannel()\n");
}

void CPPChannel::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPChannel)
    DDX_Control(pDX, IDC_CHANNEL_TREE, m_tcChannelCat);
    DDX_Control(pDX, IDC_LIST_CHANNEL, m_lcChannel);
    DDX_Text(pDX, IDC_STATIC_CHANNEL_COUNT, m_strCount);
    DDX_Text(pDX, IDC_STATIC_MESSAGE, m_strMessage);
    //DDX_Text(pDX, IDC_ST_STAGEID, m_strStageID);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPChannel, CPropertyPage)
    //{{AFX_MSG_MAP(CPPChannel)
    ON_NOTIFY(NM_CLICK, IDC_LIST_CHANNEL, OnClickListChannel)
    ON_NOTIFY(NM_DBLCLK, IDC_LIST_CHANNEL, OnDbldclkListChannel)
    ON_BN_CLICKED(IDC_BTN_RENEW, OnBtnRenew)
    ON_BN_CLICKED(IDC_BTN_MEMBER, OnBtnMember)
    ON_NOTIFY(LVN_DELETEITEM, IDC_LIST_CHANNEL, OnDeleteitemListChannel)
    ON_WM_DESTROY()
    ON_WM_ERASEBKGD()
    //ON_BN_CLICKED(IDC_BTN_MUD, OnBtnMud)
    ON_BN_CLICKED(IDC_RADIO_ENG, OnSelectEngl)
    ON_BN_CLICKED(IDC_RADIO_CHI, OnSelectChi)
    ON_BN_CLICKED(IDC_RADIO_JAP, OnSelectJap)
    ON_BN_CLICKED(IDC_RADIO_KOR, OnSelectKor)
    ON_BN_CLICKED(IDC_RADIO_OTH, OnSelectOtr)
    ON_BN_CLICKED(IDC_RADIO_SPA, OnSelectSpn)
    ON_NOTIFY(TVN_SELCHANGED, IDC_CHANNEL_TREE, OnSelchangedChannelTree)
    ON_NOTIFY(TVN_SELCHANGING, IDC_CHANNEL_TREE, OnSelchangingChannelTree)
    //}}AFX_MSG_MAP
    ON_MESSAGE(CMD_QUERY_CHANNELS, OnQueryChannels)
    ON_MESSAGE(CMD_QUERY_CHANNELS_END, OnQueryChannelsEnd)
    ON_MESSAGE(CMD_QUERY_NOMATCHES, OnQueryNoMatches)
    ON_MESSAGE(CMD_QUERY_ERROR, OnQueryError)
END_MESSAGE_MAP()

BOOL CPPChannel::OnEraseBkgnd(CDC* pDC)

```

```

{
    CPropertyPage::OnEraseBkgnd(pDC);

    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);
    return pPSJC->OnPageEraseBkgnd(pDC);
}

/*
HBRUSH CPPChannel::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);

    if (nCtlColor == CTLCOLOR_STATIC)
    {
        pDC->SetBkMode(TRANSPARENT);
        return (HBRUSH)*pPSJC->GetNullBrush();
    }
    else
    {
        HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
        return hbr;
    }
}
*/
// TRACE("CPSJoinChannel::OnSize(%d,%d,%d)\n", nType, cx, cy);

void CPPChannel::OnClickListChannel(NMHDR* pNMHDR, LRESULT* pResult)
{
    DWORD dwPos = ::GetMessagePos();
    CPoint point((int)LOWORD(dwPos), (int)HIWORD(dwPos));
    m_lcChannel.ScreenToClient(&point);

    if ((m_nIndex = m_lcChannel.HitTest(point)) != -1) {
/*
        // for previous version
        CS_PROPDATA cspd;
        PICS_PROPERTY picsProp =
(PICS_PROPERTY)m_lcChannel.GetItemData(m_nIndex);
        picsProp->HrGetProperty(&cspd, CSINDEX_PROP_CHANNEL_MODE);
        DWORD dwMode = *((DWORD*)cspd.pbData);
        BOOL bPrivate = (dwMode & CS_CHANNEL_PROTECTED);

        picsProp->HrGetProperty(&cspd, CSINDEX_PROP_TOPIC);
        CString strStage((CHAR*)cspd.pbData);
*/
        m_strStageID = m_lcChannel.GetItemText(m_nIndex, 3); // 0); //
syc 0707 // SubItem=0
        //[p2/0003chrn]title

        // syc 0707 begin
        //CString str1 = m_lcChannel.GetItemText(m_nIndex, 0);
        //CString str2 = m_lcChannel.GetItemText(m_nIndex, 3);
        //CString strTemp;
        //strTemp.Format("[%s] \\"%s\\\"", str1, str2);
        // end

```

PPChannel.cpp

```

        // m_strMessage = strTemp; // m_strStageID; // syc 0707 // syc
0715        //m_strSelectedStageID = "Selected Channel : " + strTemp; // syc
0707

        UpdateData(FALSE); // Write
//        m_btnBkg.EnableWindow();
    }
    *pResult = 0;
}

void CPPChannel::OnDbclckListChannel(NMHDR* pNMHDR, LRESULT* pResult)
{
    OnClickListChannel(pNMHDR, pResult);

    if (m_nIndex == -1)
        return;

    CPSJoinChannel* pSheet = (CPSJoinChannel*)GetParent();
    ASSERT(pSheet);
    CUC2Socket* pCS = pSheet->GetSocket();
    if (pCS && !pCS->IsQueryOK()) {
        AfxMessageBox(IDS_QUERY_LINE_BUSY);
        return;
    }
    // Get the MIC channel index of the current selected item
    CS_PROPDATA cspd;
    PICS_PROPERTY picsProp =
(PICS_PROPERTY)m_lcChannel.GetItemData(m_nIndex);
    if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_ID)))
        return;
    ASSERT(cspd.dwcb == sizeof(DWORD));
    DWORD dwChannelID = *((DWORD*)cspd.pbData);

    CMemberListDlg dlgML;
    dlgML.SetPPChannel(pSheet->GetDocument(), dwChannelID);
    dlgML.DoModal();

    *pResult = 0;
}

void CPPChannel::OnBtnRenew()
{
    VERIFY(ShowList());
}

void CPPChannel::OnBtnMember()
{
    CPSJoinChannel* pSheet = (CPSJoinChannel*)GetParent();
    ASSERT(pSheet);
    CMemberListDlg dlgML;
    dlgML.SetPPChannel(pSheet->GetDocument(), 0);
    dlgML.DoModal();
}

void CPPChannel::OnDeleteitemListChannel(NMHDR* pNMHDR, LRESULT* pResult)
{

```

```

NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
if (pNMListView->lParam)
{
    PICS_PROPERTY picsProp = (PICS_PROPERTY)pNMListView->lParam;
    picsProp->Release();
}
*pResult = 0;
}

BOOL CPPChannel::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    // Prepare List Control for Channels list
    m_lcChannel.SetImageList(&m_ilChannel, LVSIL_SMALL);

    // SATYA CHANGES START
    char* szColumn[] = {"Topic", "Background", "Members", ""};

    int nWidth[] = {120, 90, 62, 0}; // syc 0704

    LV_COLUMN lvC; // list view column structure
    lvC.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM; //
valid members
    lvC.fmt = LVCFMT_LEFT; // left-align column

    // Add the columns.
    for (int i=0; i < sizeof(nWidth)/sizeof(nWidth[0]); i++) {
        lvC.cx = nWidth[i]; // width of column in
pixels
        lvC.iSubItem = i;
        lvC.pszText = szColumn[i];
        if (m_lcChannel.InsertColumn(i, &lvC) == -1)
            return NULL;
    }
    ((CButton*)GetDlgItem(IDC_RADIO_ENG))->SetCheck(BST_CHECKED);
    m_chnCatType.LANG = 0;

    // Prepare Tree Control
    m_tcChannelCat.SetImageList(&m_ilChnCatTree, TVSIL_NORMAL);

    int nNumRootItem = gResMan.GetNumChanTree();
    for (i=0; i < nNumRootItem; i++)
    {
        CString strRootItem = gResMan.GetNameChanTree(i);
        HTREEITEM hRoot = m_tcChannelCat.InsertItem(strRootItem, 0, 1,
TVI_ROOT);
        int nNumSubItem = gResMan.GetNumSubItemChanTree(i);
        for (int j=0; j < nNumSubItem; j++)
        {
            CString strSubItem = gResMan.GetNameSubItemChanTree(i, j);
            HTREEITEM hItem = m_tcChannelCat.InsertItem(strSubItem, 0,
1, hRoot);
            DWORD dwData = (DWORD)(j + (1000 * i));
            m_tcChannelCat.SetItemData(hItem, dwData);
        }
    }
}

```

```

    }

    m_ChnCatType.CROOT = 0;
    m_ChnCatType.CCHILD = 0;

    HTREEITEM hItem, rootItem;

    rootItem = m_tcChannelCat.GetRootItem();
    hItem = rootItem;
    ASSERT(hItem);

    m_tcChannelCat.Expand(hItem, TVE_EXPAND);
    for (int iLoop = 0; hItem != NULL; iLoop++)
    {
        hItem = m_tcChannelCat.GetNextItem(hItem, TVGN_NEXT);
        m_tcChannelCat.Expand(hItem, TVE_EXPAND);
    }
    hItem = m_tcChannelCat.GetChildItem(rootItem);
    m_tcChannelCat.SelectItem(hItem);
    // Update the Room Categories tree control
    UpdateChannelCategories();
    // SATAYA CHANGES END
    return TRUE;
}

void CPPChannel::OnDestroy()
{
    CPropertyPage::OnDestroy();
}

// returns the index of the added item
int CPPChannel::AddItem(const int i, int iImage, PICS_PROPERTY picsProp,
TCHAR* psz)
{
    if (picsProp)
        picsProp->AddRef();

    LV_ITEM          lvI; // list view item structure
    ::ZeroMemory(&lvI, sizeof(lvI));
    lvI.mask          = LVIF_TEXT | LVIF_IMAGE | LVIF_PARAM | LVIF_STATE;
    lvI.state         = 0;
    lvI.stateMask     = 0;

    lvI.iItem         = i;
    lvI.iSubItem      = 0;
    // The parent window is responsible for storing the text.
    // The list view control will send an LVN_GETDISPINFO
    // when it needs the text to display.
    lvI.pszText       = psz; // syc 0705
    lvI.cchTextMax    = MIC_MAX_CHANNEL_NAME_LENGTH_MIC; // 63 // syc
0705

    // syc 0705 begin
    //char buffer[7];
    //_itoa(i, buffer, 10);

```

```

        //CHAR szText[17];
        //wsprintf(szText, "Channel : %s", buffer);
        //lvI.pszText          = szText;
        // end
        lvI.iImage            = iImage;
        lvI.lParam            = (LPARAM)picsProp;

        int iItem = m_lcChannel.InsertItem(&lvI);
        if ((iItem == -1) && picsProp)
            picsProp->Release();
        return iItem;
    }

    BOOL CPPChannel::AddChannel(PICS_PROPERTY picsProp)
    {
        ASSERT(picsProp);
        // Add the channel name..
        int i;
        if (FAILED(picsProp->HrGetPrivateData((PVOID*)&i)))
        {
            TRACE0("CPPChannel::AddChannel - HrGetPrivateData failed.\n");
            return FALSE;
        }

        CS_PROPDATA cspd;
        if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_CHANNEL_NAME)))
            return FALSE;
        TCHAR* szChannel = (TCHAR*)cspd.pbData;

        // szChannel : [p2/0002chrm]title...,[h2/],[ew0]
        ChnCatType chantype = GetChnCatType(szChannel);
        // Get the number of Users in the Channel
        if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_CHANNEL_CUSER)))
            return FALSE;
        int iCount = *((DWORD*)cspd.pbData);
        FindNoOfUsers(chantype, iCount);
        //

        // Filter display according to the channel type
        if ((gResMan.GetStageType(szChannel) == ST_HOME) || // syc 0712
            ((gResMan.GetStageType(szChannel) == (m_bMUD ? ST_MUD :
ST_PUBLIC)) &&
            (chantype.LANG == m_ChnCatType.LANG) &&
            (chantype.CROOT == m_ChnCatType.CROOT) &&
            (chantype.CCHILD == m_ChnCatType.CCHILD)))
        {
            if (FAILED(picsProp->HrGetProperty(&cspd,
CSINDEX_PROP_CHANNEL_MODE)))
                return FALSE;
            DWORD dwMode = *((DWORD*)cspd.pbData);

            if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_TOPIC)))
                return FALSE;
            CString strTopic((CHAR*)cspd.pbData); // syc 0712

```



```

char* szTopic = strTopic.GetBuffer(strTopic.GetLength() + 1); //
syc 0712
strTopic.ReleaseBuffer(); // syc 0712

if (FAILED(picsProp->HrGetProperty(&cspd,
CSINDEX_PROP_CHANNEL_CUSER)))
    return FALSE;
CHAR szCount[20];
int nCount = *((DWORD*)cspd.pbData);
wsprintf(szCount, "%d", nCount);
m_nUsers += nCount;

int iImg;
if (dwMode & CS_CHANNEL_PROTECTED)
    iImg = IL_CHANNELPRIV;
else
    iImg = m_bMUD ? IL_CHANNELMUD : IL_CHANNEL;

int iItem = AddItem(i, iImg, picsProp, szTopic);

CString strTemp(szChannel);
int nStart = strTemp.Find('/');
int nEnd = strTemp.Find(' ');
int length = strTemp.GetLength();
char* p = strTemp.GetBuffer(length + 1);
*(p + nEnd) = NULL;
p += nStart + 1;
strTemp.ReleaseBuffer();
CString strScene((char*)p);
gResMan.GetStageTitle(strScene);
// end
m_lcChannel.SetItemText(m_i, 1, strScene);
m_lcChannel.SetItemText(m_i, 2, szCount);
m_lcChannel.SetItemText(m_i, 3, szChannel);
m_i++;
return (iItem != -1);
}
return FALSE;
}

BOOL CPPChannel::ShowList()
{
    BeginWaitCursor();
    // SATYA CHANGES START
    m_strCount = "Channels : 0      Members : 0 ";
    SetDlgItemText(IDC_STATIC_CHANNEL_COUNT, m_strCount);
    // Re set the Room Categories tree control
    ReSetChannelCat();
    // SATYA CHANGES END
    CPSJoinChannel* pSheet = (CPSJoinChannel*)GetParent();
    ASSERT(pSheet);
    CUC2Socket* pCS = pSheet->GetSocket();
    if (!pCS)
        return FALSE;

```

```

m_strMessage.LoadString(IDS_RECEIVING_CHANNELS_LIST);
UpdateData(FALSE);

if (pCS->IsQueryOK())
{
    m_lcChannel.DeleteAllItems();
    m_i = m_nUsers = 0;
    // Say that now I'm the client of the socket query service (to
let CUC2Socket know)
    pCS->SetQueryClient(GetSafeHwnd());
    BeginWaitCursor();

    if (!pCS->FQueryListChannels())
        EndDialog(-1);
}
else
{
    AfxMessageBox(IDS_QUERY_LINE_BUSY);
}
// Then PSocket will call AddChannel for each item
EndWaitCursor();
return TRUE;
}

void CPPChannel::EndOfList()
{
    if (m_bMUD)
    {
        AfxFormatString1(m_strMessage, IDS_MUD_MODE, m_strStageID);
    }
    else
    {
        m_strMessage.LoadString(m_lcChannel.GetItemCount()
                                ? IDS_CLICK_CHANNEL_TITLE
                                : IDS_NO_CHANNEL_OPENED);
    }
    UpdateData(FALSE);
    UpdateChannelCategories();
    EndWaitCursor();
    // ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
}

////////////////////////////////////
// to handle messages from ChatSock

LRESULT CPPChannel::OnQueryChannels(WPARAM wParam, LPARAM lParam)
{
    // TRACE("<=0x%lx RECV\n", lParam);
    if (!AddChannel((PICS_PROPERTY)lParam))
        return -1;
    m_strCount.Format("%d:%d", m_lcChannel.GetItemCount(), m_nUsers);
    SetDlgItemText(IDC_STATIC_CHANNEL_COUNT, m_strCount); // for speed
    return 0;
}

LRESULT CPPChannel::OnQueryChannelsEnd(WPARAM wParam, LPARAM lParam)
{

```

```

        EndOfList();
        return 0;
    }

LRESULT CPPChannel::OnQueryNoMatches(WPARAM wParam, LPARAM lParam)
{
    m_strCount = "0:0";
    // m_strMessage = (char*)lParam;
    m_strMessage.LoadString(IDS_NO_CHANNEL_OPENED);
    UpdateData(FALSE); // Write
    // m_pSheet->ChatVoice(m_strMessage);
    return 0;
}

LRESULT CPPChannel::OnQueryError(WPARAM wParam, LPARAM lParam)
{
    m_strCount = "0:0";
    m_strMessage = (char*)lParam;
    m_strMessage.LoadString(IDC_STATIC_MESSAGE);
    UpdateData(FALSE); // Write
    // m_pSheet->ChatVoice(m_strMessage);
    return 0;
}

/*
void CPPChannel::OnBtnCreateChannel()
{
    m_pSheet->AddCreateChannelPage();
    GetDlgItem(IDC_BTN_CREATE_CHANNEL)->EnableWindow(FALSE);
}
*/

BOOL CPPChannel::OnApply()
{
    CPSJoinChannel* pSheet = (CPSJoinChannel*)GetParent();
    ASSERT(pSheet);
    if (m_bMUD) {
        ;
    }
    else if (!pSheet->IsListOnly() && (m_nIndex == -1) && !pSheet->IsCreateChannel()) {
        if (m_lcChannel.GetItemCount() == 0) {
            //if (m_strMessage.LoadString(IDS_NO_CHANNEL_OPENED)) //
syc 0715
            // UpdateData(FALSE); // syc 0715
            pSheet->ActivateCreateChannelPage();
            AfxMessageBox(IDS_FILL_CHANNEL_TITLE);
        }
        else {
            //if (m_strMessage.LoadString(IDS_SELECT_CHANNEL)) // syc
0715
            // UpdateData(FALSE); // syc 0715
            pSheet->ActivateJoinChannelPage();
        }
        return FALSE;
    }
    else {

```

```

        LV_ITEM it;
        ::ZeroMemory(&it, sizeof(it));          // Missing this was the bug!
        it.iItem = m_nIndex;
        it.mask = LVIF_IMAGE;
        if (m_lcChannel.GetItem(&it) && (it.iImage == IL_CHANNELPRIV)) {
            CInputPassword IPDlg;
            if (IPDlg.DoModal() == IDOK) { //&&
                !IPDlg.m_strPassword.IsEmpty()
                    m_strPassword = IPDlg.m_strPassword;
            }
            else {
                return FALSE;
            }
        }
        return CPropertyPage::OnApply();
    }

/*void CPPChannel::OnBtnMud()
{
    m_bMUD = !m_bMUD;
    // GetDlgItem(IDC_LIST_CHANNEL)->EnableWindow(!m_bMUD);
    if (m_bMUD)
    {
        m_nIndex = -1;
        m_strPassword.Empty();
        GetLastStageID();
    }
    else
    {
        m_strStageID.Empty();
    }
    SetMUDButtonTitle();
    OnBtnRenew();
}*/

void CPPChannel::GetLastStageID()
{
    ((CUC2App*)AfxGetApp())->RegGetLastStageID(m_strStageID);
    if (m_strStageID.IsEmpty())
    {
        m_strStageID = *gResMan.GetStageName(0);
        gResMan.ExtractStageID(m_strStageID);
    }
    gResMan.MakeStageName(m_strStageID, FALSE);    // MUD name
}

// SATYA CHANGES START
// removed MUD Button we don't need it
/*void CPPChannel::SetMUDButtonTitle()
{
    CString strBtn;
    strBtn.LoadString(m_bMUD ? IDS_BTN_MUD : IDS_BTN_USER);
    GetDlgItem(IDC_BTN_MUD)->SetWindowText(strBtn);
}
*/
//Function Name : GetChnCatType

```

PPChannel.cpp

```
//Parameter      : Room Name
//Purpose        : This function Parse the room name and find out the
//                the room categoriy (ENGLISH,AGE,20+)
//Return         : Return the Room Categoriy.
```

```
ChnCatType CPPChannel::GetChnCatType(LPCTSTR szName) const
```

```
{
    ChnCatType chantype;
    chantype.LANG = 0;
    chantype.CROOT = 0;
    chantype.CCHILD = 0;

    char *channel_name = (char *)szName;
    gParser.CopyBuffer(channel_name);
    gParser.SetLeftToken(' ');
    gParser.GetValueRightToken(chantype.LANG, ' ');
    gParser.GetValueRightToken(chantype.CROOT, ' ');
    gParser.GetValueRightToken(chantype.CCHILD, ' ');

    return chantype;
}
```

```
// When select English Radio Button
```

```
void CPPChannel::OnSelectEngl()
```

```
{
    m_ChnCatType.LANG = 0;
    VERIFY>ShowList();
}
```

```
// When select Chinese Radio Button
```

```
void CPPChannel::OnSelectChi()
```

```
{
    m_ChnCatType.LANG = 3;
    VERIFY>ShowList();
}
```

```
// When select Japanese Radio Button
```

```
void CPPChannel::OnSelectJap()
```

```
{
    m_ChnCatType.LANG = 4;
    VERIFY>ShowList();
}
```

```
// When select Korean Radio Button
```

```
void CPPChannel::OnSelectKor()
```

```
{
    m_ChnCatType.LANG = 2;
    VERIFY>ShowList();
}
```

```
// When select Others Radio Button
```

```
void CPPChannel::OnSelectOtr()
```

```
{
```

PPChannel.cpp

```

        m_ChnCattType.LANG = 5;
        VERIFY(ShowList());
    }
    // When select Spanish Radio Button
    void CPPChannel::OnSelectSpn()
    {
        m_ChnCattType.LANG = 1;
        VERIFY(ShowList());
    }

    // On Channel Tree Selection Changed
    void CPPChannel::OnSelchangedChannelTree(NMHDR* pNMHDR, LRESULT* pResult)
    {
        NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
        HTREEITEM hItem = pNMTreeView->itemNew.hItem;
        if (m_tcChannelCat.ItemHasChildren(hItem))
        {
            m_tcChannelCat.Expand(hItem, TVE_EXPAND);
            m_lcChannel.DeleteAllItems();
            return;
        }

        VERIFY(ShowList());
        *pResult = 0;
    }
    // On before Channel Tree Selection changing
    void CPPChannel::OnSelchangingChannelTree(NMHDR* pNMHDR, LRESULT* pResult)
    {
        NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
        HTREEITEM hItem = pNMTreeView->itemNew.hItem;

        if (m_tcChannelCat.ItemHasChildren(hItem))
            return;
        DWORD dwData = m_tcChannelCat.GetItemData(hItem);
        m_ChnCattType.CROOT = (int)(dwData / 1000);
        m_ChnCattType.CCHILD = (int)(dwData % 1000);

        *pResult = 0;
    }

    // Function Name : UpdateChannelCategories
    // Parameters    : None
    // Purpose       : It updates the room categories tree control with No of
    Users
    // Return        : TRUE if it Updates the tree control else FALSE

    BOOL CPPChannel::UpdateChannelCategories()
    {
        CString ItemText,tempstr;
        int iIndex = 0;
        HTREEITEM hitem,prItem;
    }

```

```

// Update the frist node

prItem = m_tcChannelCat.GetRootItem();
hitem = prItem;
ASSERT(hitem);
ItemText = m_tcChannelCat.GetItemText(hitem);
tempstr.Format("(%d)", m_aChatCentral[0]);
iIndex = ItemText.Find('(');
if (iIndex != -1)
{
    // delete the no of People in the Room categoiry..
    ItemText.Delete(iIndex, (ItemText.GetLength() - iIndex));
}
ItemText += tempstr;
m_tcChannelCat.SetItemText(hitem, LPCTSTR(ItemText));

hitem = m_tcChannelCat.GetChildItem(hitem);
ASSERT(hitem);
ItemText = m_tcChannelCat.GetItemText(hitem);
iIndex = ItemText.Find('(');
if (iIndex != -1)
{
    ItemText.Delete(iIndex, (ItemText.GetLength() - iIndex));
}
ItemText += tempstr;
m_tcChannelCat.SetItemText(hitem, LPCTSTR(ItemText));

// update the secoend node

prItem = m_tcChannelCat.GetNextItem(prItem, TVGN_NEXT);
hitem = prItem;
ASSERT(hitem);
ItemText.Empty();
tempstr.Empty();

ItemText = m_tcChannelCat.GetItemText(hitem);
tempstr.Format("(%d)", m_aAge[0] + m_aAge[1] + m_aAge[2] + m_aAge[3] + m_aAge[4]
);
iIndex = ItemText.Find('(');
if (iIndex != -1)
{
    ItemText.Delete(iIndex, (ItemText.GetLength() - iIndex));
}
ItemText += tempstr;
m_tcChannelCat.SetItemText(hitem, LPCTSTR(ItemText));

// Get the Childeran
hitem = m_tcChannelCat.GetChildItem(prItem);

for (int iLoop = 0; iLoop < 5; iLoop++)
{
    ItemText.Empty();
    tempstr.Empty();

```

```

        ItemText = m_tcChannelCat.GetItemText(hitem);
        tempstr.Format("(%d",m_aAge[iLoop]);
        iIndex = ItemText.Find('(');
        if (iIndex !=-1)
        {
            ItemText.Delete(iIndex, (ItemText.GetLength()-iIndex));
        }
        ItemText += tempstr;
        m_tcChannelCat.SetItemText(hitem,LPCTSTR(ItemText));
        hitem = m_tcChannelCat.GetNextItem(hitem,TVGN_NEXT);
    }

    // update for the third node

    ItemText.Empty();
    tempstr.Empty();
    // get the parent node
    prItem = m_tcChannelCat.GetNextItem(prItem,TVGN_NEXT);
    hitem=prItem;
    ItemText = m_tcChannelCat.GetItemText(hitem);
    tempstr.Format("(%d",m_aEthic[0]+m_aEthic[1]+m_aEthic[2]+m_aEthic[3]);
    iIndex = ItemText.Find('(');
    if (iIndex !=-1)
    {
        ItemText.Delete(iIndex, (ItemText.GetLength()-iIndex));
    }
    ItemText += tempstr;
    m_tcChannelCat.SetItemText(hitem,LPCTSTR(ItemText));

    // Get the child nodes
    hitem = m_tcChannelCat.GetChildItem(hitem);

    for (iLoop =0;iLoop<4;iLoop++)
    {
        ItemText.Empty();
        tempstr.Empty();

        ItemText = m_tcChannelCat.GetItemText(hitem);
        tempstr.Format("(%d",m_aEthic[iLoop]);
        iIndex = ItemText.Find('(');
        if (iIndex !=-1)
        {
            ItemText.Delete(iIndex, (ItemText.GetLength()-iIndex));
        }
        ItemText += tempstr;
        m_tcChannelCat.SetItemText(hitem,LPCTSTR(ItemText));
        hitem = m_tcChannelCat.GetNextItem(hitem,TVGN_NEXT);
    }

    // update the fourth node;

    ItemText.Empty();
    tempstr.Empty();
    // Get the parent Node
    prItem = m_tcChannelCat.GetNextItem(prItem,TVGN_NEXT);
    hitem = prItem;

    ItemText = m_tcChannelCat.GetItemText(hitem);

```



```

        tempstr.Format("(%d)", m_aRegional[0]+m_aRegional[1]+m_aRegional[2]+m_aRegional[3]);
        iIndex = ItemText.Find('(');
        if (iIndex != -1)
        {
            ItemText.Delete(iIndex, (ItemText.GetLength()-iIndex));
        }
        ItemText += tempstr;
        m_tcChannelCat.SetItemText(hitem, LPCTSTR(ItemText));

        hitem = m_tcChannelCat.GetChildItem(hitem);
        for (iLoop = 0; iLoop < 4; iLoop++)
        {
            ItemText.Empty();
            tempstr.Empty();

            ItemText = m_tcChannelCat.GetItemText(hitem);
            tempstr.Format("(%d)", m_aRegional[iLoop]);
            iIndex = ItemText.Find('(');
            if (iIndex != -1)
            {
                ItemText.Delete(iIndex, (ItemText.GetLength()-iIndex));
            }
            ItemText += tempstr;
            m_tcChannelCat.SetItemText(hitem, LPCTSTR(ItemText));
            hitem = m_tcChannelCat.GetNextItem(hitem, TVGN_NEXT);
        }

        // update the last node

        ItemText.Empty();
        tempstr.Empty();

        // Get the parent node
        prItem = m_tcChannelCat.GetNextItem(prItem, TVGN_NEXT);
        hitem = prItem;
        ItemText = m_tcChannelCat.GetItemText(hitem);
        tempstr.Format("(%d)", m_intrests[0]+m_intrests[1]+m_intrests[2]+m_intrests[3]+m_intrests[4]+m_intrests[5]+m_intrests[6]);
        iIndex = ItemText.Find('(');
        if (iIndex != -1)
        {
            ItemText.Delete(iIndex, (ItemText.GetLength()-iIndex));
        }
        ItemText += tempstr;
        m_tcChannelCat.SetItemText(hitem, LPCTSTR(ItemText));

        // get the child node
        hitem = m_tcChannelCat.GetChildItem(hitem);
        for (iLoop = 0; iLoop < 7; iLoop++)
        {
            ItemText.Empty();
            tempstr.Empty();

```

```

        ItemText = m_tcChannelCat.GetItemText(hitem);
        tempstr.Format("(%d)",m_intrests[iLoop]);
        iIndex = ItemText.Find('(');
        if (iIndex !=-1)
        {
            ItemText.Delete(iIndex, (ItemText.GetLength()-iIndex));
        }
        ItemText += tempstr;
        m_tcChannelCat.SetItemText(hitem,LPCTSTR(ItemText));
        hitem = m_tcChannelCat.GetNextItem(hitem,TVGN_NEXT);
    }
    return TRUE;
}

// Function Name : FindNoOfUsers
// Parameters    : Channel type and count
// Purpose       : This function calculates the No Of Users
// Return        : void

void CPPChannel::FindNoOfUsers(ChnCatType &chantype, int iCount)
{
    if(chantype.LANG ==m_ChnCatType.LANG )
    {
        if (chantype.CROOT == 0)
        {
            m_aChatCentral[chantype.CCHILD] += iCount;
        }
        else if (chantype.CROOT == 1)
        {
            m_aAge[chantype.CCHILD] += iCount;
        }
        else if (chantype.CROOT == 2)
        {
            m_aEthic[chantype.CCHILD] += iCount;
        }
        else if (chantype.CROOT == 3)
        {
            m_aRegional[chantype.CCHILD] += iCount;
        }
        else if (chantype.CROOT == 4)
        {
            m_intrests[chantype.CCHILD] += iCount;
        }
    }
}

// Function Name : ReSetChannalCat
// Parameters     : None
// Purpose        : This function reset all the Room Categori arrays
// Return         : None
void CPPChannel::ReSetChannalCat()
{
    //inititalize all the arrays to 0
    // I am too crazy while writting this code...

    for (int iLoop =0;iLoop < 7;iLoop++)

```

```
{
    switch (iLoop)
    {
    case 0:
        m_aChatCentral[iLoop] = 0;
        m_aAge[iLoop] = 0;
        m_aEthic[iLoop] = 0;
        m_aRegional[iLoop] = 0;
        m_intrests[iLoop] = 0;
        break;
    case 1:
    case 2:
    case 3:
        m_aAge[iLoop] = 0;
        m_aEthic[iLoop] = 0;
        m_aRegional[iLoop] = 0;
        m_intrests[iLoop] = 0;
        break;
    case 4:
        m_aAge[iLoop] = 0;
        m_intrests[iLoop] = 0;
        break;
    case 5:
    case 6:
        m_intrests[iLoop] = 0;
        break;
    };
}
// SATYA CHANGES ENDS
```

PPChannel.h

```
// PPChannel.h : header file
//
//=====
//      (C) Programmed by Kim,
//      Information Technology Institute
//      UNICHAT.COM
//=====

#ifndef __PPCHANNEL_H__
#define __PPCHANNEL_H__

////////////////////////////////////
// CPPChannel dialog

// SATYA CHANGES START
// this struct will maintain the Room Combination
// Language/Room Categories (Main and SubItem)
typedef struct tagChnCatType
{
    int LANG;
    int CROOT;
    int CCHILD;
} ChnCatType;
// SATYA CHANGES END

class CPPChannel : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPChannel)

// Construction
public:
    CPPChannel();
    ~CPPChannel();

    enum CHANNEL_IL_INDEX
    {
        IL_CHANNEL=0,
        IL_CHANNELPRIV,
        IL_CHANNELMUD,
        IL_COUNT
    };

    int          AddItem(const int i, int iImage, PICS_PROPERTY picsProp,
TCHAR* psz);
    BOOL  AddChannel(PICS_PROPERTY picsProp);
    BOOL  ShowList();
    void  EndOfList();
    BOOL  IsMUD() const      { return m_bMUD; }
    void  GetLastStageID();

    CString m_strPassword;

// Dialog Data
    //{AFX_DATA(CPPChannel)
    enum { IDD = IDD_PP_CHANNEL };
    CTreeCtrl  m_tcChannelCat;
};
```

PPChannel.h

```

CListCtrl    m_lcChannel;
CString      m_strCount;
CString      m_strMessage;
CString      m_strStageID;
//}}AFX_DATA

// Overrides
// ClassWizard generate virtual function overrides
//{{AFX_VIRTUAL(CPPChannel)
protected:
virtual BOOL OnApply();
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//void        SetMUDButtonTitle(); //NOMUDBTN001
// SATYA CHANGES START
// This function return the Room Categories
ChnCatType GetChnCatType(LPCTSTR szName) const;
// SATYA CHANGES END

    int                m_nIndex;    // Current selection index for
ListControl
    int                m_nUsers;    // Total users
    int                m_i;         // item index
    CFont              m_font;
    CImageList         m_ilChannel;
    CImageList         m_ilChnCatTree;
    BOOL               m_bMUD;      // MUD mode
// SATYA CHANGES START
ChnCatType m_ChnCatType;
// following array added to maintain the no of members online.
int m_aChatCentral[1];
int m_aAge[5];
int m_aEthnic[4];
int m_aRegional[4];
int m_intrests[7];
// SATYA CHANGES END

// Generated message map functions
//{{AFX_MSG(CPPChannel)
afx_msg void OnClickListChannel(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnDblclkListChannel(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnBtnRenew();
afx_msg void OnBtnMember();
afx_msg void OnDeleteitemListChannel(NMHDR* pNMHDR, LRESULT* pResult);
virtual BOOL OnInitDialog();
afx_msg void OnDestroy();
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
//afx_msg void OnBtnMud(); // NOMUDBTN001
afx_msg void OnSelectEngl();
afx_msg void OnSelectChi();
afx_msg void OnSelectJap();
afx_msg void OnSelectKor();

```

PPChannel.h

```
afx_msg void OnSelectOtr();
afx_msg void OnSelectSpn();
afx_msg void OnSelchangedChannelTree(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnSelchangingChannelTree(NMHDR* pNMHDR, LRESULT* pResult);
//}}AFX_MSG
afx_msg LRESULT OnQueryChannels(WPARAM, LPARAM);
afx_msg LRESULT OnQueryChannelsEnd(WPARAM, LPARAM);
afx_msg LRESULT OnQueryNoMatches(WPARAM, LPARAM);
afx_msg LRESULT OnQueryError(WPARAM, LPARAM);
DECLARE_MESSAGE_MAP()

private:
    // SATYA CHANGES START
    // Reset the Room Categories
    void ReSetChannalCat();
    // Find the total number users online
    void FindNoOfUsers(ChnCatType& chantype,int iCount);
    // Updates the Room Categories tree control..
    BOOL UpdateChannelCategories();
    // SATYA CHANGES END
};
#endif // __PPCHANNEL_H__
```

```

// PPSCreateChannel.cpp : implementation file
//
//=====
//      (C) Programmed by Kim, Soomin, Mar 1998
//      Information Technology Institute
//      UNICHAT NETWORKS INC
//=====

#include "stdafx.h"
#include "resource.h"
#include "PPCreateChannel.h"
#include "PSJoinChannel.h"
#include "ResMan.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

IMPLEMENT_DYNCREATE(CPPCreateChannel, CPropertyPage)

////////////////////////////////////
// CPPCreateChannel property page

CPPCreateChannel::CPPCreateChannel() : CPropertyPage(CPPCreateChannel::IDD)
{
    TRACE0("CPPCreateChannel::CPPCreateChannel()\n");
    //{{AFX_DATA_INIT(CPPCreateChannel)
    m_strPassword = _T("");
    m_strTopic = _T("");
    //}}AFX_DATA_INIT
    m_bPublic = TRUE;
    // SATYA CHANGES START
    // load the images for the Tree control
    m_ilChanCatTree.Create(IDB_IL_CREATECHANNEL, 16, 1, CLR_NONE);
    for (int i=0; i < 3; i++) // syc 0628
        m_aChanCatType[i] = 0;
    // SATYA CHANGES END
}

CPPCreateChannel::~CPPCreateChannel()
{
    TRACE0("CPPCreateChannel::~CPPCreateChannel()\n");
}

void CPPCreateChannel::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPCreateChannel)
    DDX_Control(pDX, IDC_CREATECHANNEL_TREE, m_clChanCatTree);
    DDX_Control(pDX, IDC_LB_BACKGROUND, m_lbBackground);
    DDX_Text(pDX, IDC_EDIT_PASSWORD, m_strPassword);
    DDV_MaxChars(pDX, m_strPassword, 12);
    DDX_Text(pDX, IDC_EDIT_CHANNEL_NAME, m_strTopic);
    DDV_MaxChars(pDX, m_strTopic, 100);
    //}}AFX_DATA_MAP
}

```

```

        //}}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CPPCreateChannel, CPropertyPage)
    //{{AFX_MSG_MAP(CPPCreateChannel)
    ON_WM_ERASEBKGD()
    ON_BN_CLICKED(IDC_RADIO_PRIVATE, OnRadioPrivate)
    ON_BN_CLICKED(IDC_RADIO_PUBLIC, OnRadioPublic)
    ON_LBN_SELCHANGE(IDC_LB_BACKGROUND, OnSelchangeLbBackground)
    ON_BN_CLICKED(IDC_RAD_CRCNL_CHI, OnSelectChinese)
    ON_BN_CLICKED(IDC_RAD_CRCNL_ENG, OnSelectEnglish)
    ON_BN_CLICKED(IDC_RAD_CRCNL_JAP, OnSelectJapanese)
    ON_BN_CLICKED(IDC_RAD_CRCNL_KOR, OnSelectKorean)
    ON_BN_CLICKED(IDC_RAD_CRCNL_OTH, OnSelectOthers)
    ON_BN_CLICKED(IDC_RAD_CRCNL_SPA, OnSelectSpanish)
    ON_NOTIFY(TVN_SELCHANGED, IDC_CREATCHANNEL_TREE, OnSelectChnCatTree)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CPPCreateChannel::OnEraseBkgnd(CDC* pDC)
{
    CPropertyPage::OnEraseBkgnd(pDC);

    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);
    return pPSJC->OnPageEraseBkgnd(pDC);
}
/*
HBRUSH CPPCreateChannel::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);

    if (nCtlColor == CTLCOLOR_STATIC)
    {
        pDC->SetBkMode(TRANSPARENT);
        return (HBRUSH)*pPSJC->GetNullBrush();
    }
    else
    {
        HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
        return hbr;
    }
}
*/
void CPPCreateChannel::OnRadioPrivate()
{
    m_bPublic = FALSE;
    CEdit* pEdit = (CEdit*)GetDlgItem(IDC_EDIT_PASSWORD);
    pEdit->ShowWindow(SW_SHOW);
    CStatic* pPassSta = (CStatic*)GetDlgItem(IDC_STATIC_PASSWORD);

    pPassSta->ShowWindow(SW_SHOW);
    //pEdit->EnableWindow();
    pEdit->SetFocus();
}

```



```

}

void CPPCreateChannel::OnRadioPublic()
{
    m_bPublic = TRUE;

    //GetDlgItem(IDC_EDIT_PASSWORD)->EnableWindow(FALSE);
    CEdit* pEdit = (CEdit*)GetDlgItem(IDC_EDIT_PASSWORD);
    pEdit->ShowWindow(SW_HIDE);
    CStatic *pPassSta = (CStatic*)GetDlgItem(IDC_STATIC_PASSWORD);
    pPassSta->ShowWindow(SW_HIDE);
    GetDlgItem(IDC_EDIT_CHANNEL_NAME)->SetFocus();
}

void CPPCreateChannel::OnSelchangeLbBackground()
{
    /*m_lbBackground.GetText(m_lbBackground.GetCurSel(), m_strStageName);
    // gResMan.MakeStageName(m_strStageName, TRUE);    // Public
    GetDlgItem(IDC_EDIT_CHANNEL_NAME)->SetFocus();*/
    CString* pS = gResMan.GetStageName(m_lbBackground.GetCurSel() + 1);
    m_strStageName = *pS;

    GetDlgItem(IDC_EDIT_CHANNEL_NAME)->SetFocus();
}

BOOL CPPCreateChannel::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);
    GetDlgItem(IDC_EDIT_CHANNEL_NAME)->SetFont(pPSJC->GetFont(), FALSE);

    ((CButton*)GetDlgItem(IDC_RADIO_PUBLIC))->SetCheck(BST_CHECKED);
    OnRadioPublic();

    for (int i=1; i < gResMan.GetNumStageNames(); i++)
    {
        CString* pS = gResMan.GetStageName(i);
        if (pS)
        {
            // syc 0705 begin
            CString strScene = *pS;
            int nStart = strScene.Find(' ');
            int length = strScene.GetLength();
            char* p = strScene.GetBuffer(length + 1);
            p += nStart + 1;
            CString strTemp((char*)p);
            strScene.ReleaseBuffer();
            strScene = strTemp;
            // end
            m_lbBackground.AddString(strScene);    // (*pS); // syc 0705
        }
    }
    m_lbBackground.SetCurSel(0);
    OnSelchangeLbBackground();
    // SATYA CHANGES START

```

```

// create the Tree control and initialize the tree control
// with Image list for Tree control
m_clChanCatTree.SetImageList(&m_ilChanCatTree, TVSIL_NORMAL);

int nNumRootItem = gResMan.GetNumChanTree();
for (i=0; i < nNumRootItem; i++)
{
    CString strRootItem = gResMan.GetNameChanTree(i);
    HTREEITEM hRoot = m_clChanCatTree.InsertItem(strRootItem, 0, 1,
TVI_ROOT);
    int nNumSubItem = gResMan.GetNumSubItemChanTree(i);
    for (int j=0; j < nNumSubItem; j++)
    {
        CString strSubItem = gResMan.GetNameSubItemChanTree(i, j);
        HTREEITEM hItem = m_clChanCatTree.InsertItem(strSubItem,
2,3, hRoot);
        DWORD dwData = (DWORD)(j + (1000 * i));
        m_clChanCatTree.SetItemData(hItem, dwData);
    }
}

HTREEITEM hItem = m_clChanCatTree.GetRootItem();
if (hItem)
{
    // Select the Default item
    m_clChanCatTree.SelectItem(hItem);
}

m_aChanCatType[1] = 0;
m_aChanCatType[2] = 0;

// Set English as the Default Language
((CButton*)GetDlgItem(IDC_RAD_CRCNL_ENG))->SetCheck(BST_CHECKED);
OnSelectEnglish();
// SATYA CHANGES END

return TRUE; // return TRUE unless you set the focus to a control
// EXCEPTION: OCX Property Pages should return FALSE
}

BOOL CPPCreateChannel::OnApply()
{
    // SATYA CHANGES START

    CPSJoinChannel *pParent = NULL;
    pParent = (CPSJoinChannel*) GetParent();
    if (pParent)
    {
        CPropertyPage *Current = pParent->GetActivePage();
        // if CreateChannel is the current Active Page
        if (Current == this)
        {
            HTREEITEM hItem =m_clChanCatTree.GetSelectedItem();
            if(hItem)

```

```

        {
            // if user does not select any Room Category
            // then give a Message to select the Room category.
            if (m_clChanCatTree.ItemHasChildren(hItem))
            {
                AfxMessageBox("Please select Channl
Category");
                GetDlgItem(IDC_CREATCHANNEL_TREE)->SetFocus();
                return FALSE;
            }
        }
        if(m_strTopic.IsEmpty())
        {
            AfxMessageBox("Please enter the Topic Name");
            return FALSE;
        }
    }
    // SATYA CHANGES END

    if (!IsPublic() && m_strPassword.IsEmpty())
    {
        AfxMessageBox(IDS_ENTER_PASSWORD);
        GetDlgItem(IDC_EDIT_PASSWORD)->SetFocus();
        return FALSE;
    }

    if (m_strStageName.IsEmpty())
    {
        TRACE0("CPPCreateChannel::OnApply() - m_strStageName is
Empty!\n");
        AfxMessageBox(IDS_SELECT_STAGE);
        GetDlgItem(IDC_LB_BACKGROUND)->SetFocus();
        return FALSE;
    }
    // SATYA CHANGES START

    if (!m_strTopic.IsEmpty())
    {
        gResMan.ExtractStageID(m_strStageName);
        gResMan.MakeStageName(m_strStageName, TRUE);
        CString str;
        str.Format("(%d,%d,%d)", m_aChanCatType[0], m_aChanCatType[1],
m_aChanCatType[2]);
        m_strStageName += str;
        // m_strStageName : [p2/0001ctrm]{0,0,0}
        // Set the current selction so so that
        // we can get the Dynamic Banner..
        gResMan.SetRoomCatigories(m_aChanCatType[1],m_aChanCatType[2]);

        // SATYA CHANGES END
        m_strStageName += m_strTopic;
    }
    // SATYA CHANGES START

```

```

        // before creating the room store the Room Categories

        // SATYA CHANGES END
        return CPropertyPage::OnApply();
    }
    // SATYA CHANGES SATART
    // Onselect Chinese Radio Button
    void CPPCreateChannel::OnSelectChinese()
    {
        m_aChanCatType[0] = 3;
    }
    // Onselect English Radio Button
    void CPPCreateChannel::OnSelectEnglish()
    {
        m_aChanCatType[0] = 0;
    }
    // Onselect Japanese Radio Button
    void CPPCreateChannel::OnSelectJapanese()
    {
        m_aChanCatType[0] = 4;
    }
    // Onselect Korean Radio Button
    void CPPCreateChannel::OnSelectKorean()
    {
        m_aChanCatType[0] = 2;
    }
    // Onselect Others Radio Button
    void CPPCreateChannel::OnSelectOthers()
    {
        m_aChanCatType[0] = 5;
    }
    // Onselect Spanish Radio Button
    void CPPCreateChannel::OnSelectSpanish()
    {
        m_aChanCatType[0] = 1;
    }
    // On select Room Category Tree control
    void CPPCreateChannel::OnSelectChnCatTree(NMHDR* pNMHDR, LRESULT* pResult)
    {
        NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
        HTREEITEM hItem = pNMTreeView->itemNew.hItem;

        if (m_clChanCatTree.ItemHasChildren(hItem))
        {
            m_clChanCatTree.Expand(hItem, TVE_EXPAND);

            return;
        }

        DWORD dwData = m_clChanCatTree.GetItemData(hItem);
    }

```

PPCreateChannel.cpp

```
    m_aChanCatType[1] = (int)(dwData / 1000);  
    m_aChanCatType[2] = (int)(dwData % 1000);  
  
    *pResult = 0;  
}  
// SATYA CHANGES END
```

PPCreateChannel.h

```
// PPSCreateChannel.h : header file
//
//=====
//      (C) Programmed by Kim, Mar 1998
//      Information Technology Institute
//      UNICHAT.COM
//=====

#ifndef __PPCREATECHANNEL_H__
#define __PPCREATECHANNEL_H__

////////////////////////////////////
// CPPCreateChannel dialog

class CPPCreateChannel : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPCreateChannel)

// Construction
public:
    CPPCreateChannel();
    ~CPPCreateChannel();

    BOOL        IsPublic() const        { return m_bPublic; }
    CString*    GetStageName()          { return &m_strStageName; }

// Dialog Data
   //{{AFX_DATA(CPPCreateChannel)
    enum { IDD = IDD_PP_CREATE_CHANNEL };
    CTreeCtrl    m_clChanCatTree;
    CListBox     m_lbBackground;
    CString      m_strPassword;
    CString      m_strTopic;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPPCreateChannel)
    public:
    virtual BOOL OnApply();
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    BOOL        m_bPublic;
    CString      m_strStageName;    // resource name [p2/0000abcd]
    // SATYA CHANGES START
    CImageList    m_ilChanCatTree; // Image list for Tree control
    int           m_aChanCatType[3]; // Maintain the Room Category
    // SATYA CHANGES END

    // Generated message map functions
    //{{AFX_MSG(CPPCreateChannel)
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);

```

PPCreateChannel.h

```
afx_msg void OnRadioPrivate();
afx_msg void OnRadioPublic();
afx_msg void OnSelchangeLbBackground();
virtual BOOL OnInitDialog();
afx_msg void OnSelectChinese();
afx_msg void OnSelectEnglish();
afx_msg void OnSelectJapanese();
afx_msg void OnSelectKorean();
afx_msg void OnSelectOthers();
afx_msg void OnSelectSpanish();
afx_msg void OnSelectChnCatTree(NMHDR* pNMHDR, LRESULT* pResult);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

private:
};

#endif // __PPCREATECHANNEL_H__
```

PPMemberInfo1.cpp

```
// PPMemberInfo1.cpp : implementation file
//

#include "stdafx.h"
#include "resource.h"
#include "PPMemberInfo1.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CPPMemberInfo1, CPropertyPage)
IMPLEMENT_DYNCREATE(CPPMemberInfo2, CPropertyPage)

////////////////////////////////////
// CPPMemberInfo1 property page

CPPMemberInfo1::CPPMemberInfo1() : CPropertyPage(CPPMemberInfo1::IDD)
{
    //{{AFX_DATA_INIT(CPPMemberInfo1)
    m_strUserID = _T("");
    m_strVersion = _T("");
    m_strSexAge = _T("");
    //}}AFX_DATA_INIT
}

CPPMemberInfo1::~CPPMemberInfo1()
{
}

void CPPMemberInfo1::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPMemberInfo1)
    DDX_Text(pDX, IDC_STATIC_NICK, m_strNick);
    DDX_Text(pDX, IDC_STATIC_REALNAME, m_strRealName);
    DDX_Text(pDX, IDC_STATIC_UNITEL_ID, m_strUserID);
    DDX_Text(pDX, IDC_STATIC_VERSION, m_strVersion);
    DDX_Text(pDX, IDC_STATIC_SEXAGE, m_strSexAge);
    //}}AFX_DATA_MAP
}

BOOL CPPMemberInfo1::OnSetActive()
{
    UpdateData(FALSE);          // Update dialog items

    return CPropertyPage::OnSetActive();
}

BEGIN_MESSAGE_MAP(CPPMemberInfo1, CPropertyPage)
    //{{AFX_MSG_MAP(CPPMemberInfo1)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```



```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CPPMemberInfo2 property page

```

```

CPPMemberInfo2::CPPMemberInfo2() : CPropertyPage(CPPMemberInfo2::IDD)
{
    //{AFX_DATA_INIT(CPPMemberInfo2)
    m_strProfile = _T("");
    //}AFX_DATA_INIT
}

```

```

CPPMemberInfo2::~CPPMemberInfo2()
{
}

```

```

void CPPMemberInfo2::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPPMemberInfo2)
    DDX_Text(pDX, IDC_EDIT_PROFILE, m_strProfile);
    //}AFX_DATA_MAP
}

```

```

BEGIN_MESSAGE_MAP(CPPMemberInfo2, CPropertyPage)
    //{AFX_MSG_MAP(CPPMemberInfo2)
    // NOTE: the ClassWizard will add message map macros here
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

BOOL CPPMemberInfo2::OnSetActive()
{
    UpdateData(FALSE);    // Update dialog items

    return CPropertyPage::OnSetActive();
}

```

PPMemberInfo1.h

```
// PPMemberInfo1.h : header file
//
```

```
#ifndef __PPMEMBERINFO1_H__
#define __PPMEMBERINFO1_H__
```

```
#include "resource.h"
```

```
////////////////////////////////////
// CPPMemberInfo1 dialog
```

```
class CPPMemberInfo1 : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPMemberInfo1)
```

```
// Construction
```

```
public:
    CPPMemberInfo1();
    ~CPPMemberInfo1();
```

```
// Dialog Data
```

```
//{{AFX_DATA(CPPMemberInfo1)
enum { IDD = IDD_PP_MEMBER1 };
CString      m_strNick;
CString      m_strRealName;
CString      m_strUserID;
CString      m_strVersion;
CString      m_strSexAge;
//}}AFX_DATA
```

```
// Overrides
```

```
// ClassWizard generate virtual function overrides
//{{AFX_VIRTUAL(CPPMemberInfo1)
public:
    virtual BOOL OnSetActive();
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL
```

```
// Implementation
```

```
protected:
    // Generated message map functions
    {{{AFX_MSG(CPPMemberInfo1)
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

```
};
```

```
////////////////////////////////////
// CPPMemberInfo2 dialog
```

```
class CPPMemberInfo2 : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPMemberInfo2)
```

```
// Construction
```

```
public:
```

PPMemberInfo1.h

```
CPPMemberInfo2();
~CPPMemberInfo2();

// Dialog Data
//{{AFX_DATA(CPPMemberInfo2)
enum { IDD = IDD_PP_MEMBER2 };
CString      m_strProfile;
//}}AFX_DATA

// Overrides
// ClassWizard generate virtual function overrides
//{{AFX_VIRTUAL(CPPMemberInfo2)
public:
virtual BOOL OnSetActive();
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
// Generated message map functions
//{{AFX_MSG(CPPMemberInfo2)
// NOTE: the ClassWizard will add member functions here
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

};

#endif // __PPMEMBERINFO1_H__
```

PPMyAddress.cpp

```
// PPMYAddress.cpp : implementation file
//

#include "stdafx.h"
#include "uc2.h"
#include "PPMyAddress.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CPPMyAddress dialog

CPPMyAddress::CPPMyAddress(CWnd* pParent /*=NULL*/)
: CDialog(CPPMyAddress::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPPMyAddress)
    m_sCity = _T("");
    m_sState = _T("");
    m_sAddress = _T("");
    m_sZip = _T("");
    m_sTelHome = _T("");
    m_sTelMobile = _T("");
    m_sTelOther = _T("");
    m_sTelWork = _T("");
    m_sTelFax = _T("");
    //}}AFX_DATA_INIT
}

void CPPMyAddress::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPMyAddress)
    DDX_Control(pDX, IDC_CB_COUNTRY, m_cbCountry);
    DDX_Text(pDX, IDC_EDIT_ADDR_CITY, m_sCity);
    DDX_Text(pDX, IDC_EDIT_ADDR_STATE, m_sState);
    DDX_Text(pDX, IDC_EDIT_ADDR_STREET, m_sAddress);
    DDX_Text(pDX, IDC_EDIT_ADDR_ZIP, m_sZip);
    DDX_Text(pDX, IDC_EDIT_TEL_HOME, m_sTelHome);
    DDX_Text(pDX, IDC_EDIT_TEL_MOBILE, m_sTelMobile);
    DDX_Text(pDX, IDC_EDIT_TEL_OTHER, m_sTelOther);
    DDX_Text(pDX, IDC_EDIT_TEL_WORK, m_sTelWork);
    DDX_Text(pDX, IDC_EDIT_TEL_FAX, m_sTelFax);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPMyAddress, CDialog)
    //{{AFX_MSG_MAP(CPPMyAddress)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////  
// CPPMyAddress message handlers
```

PPMyAddress.h

```

#if
!defined(AFX_PPMYADDRESS_H__0E8E8442_B25D_11D3_B82B_00105A60F930__INCLUDED_)
#define AFX_PPMYADDRESS_H__0E8E8442_B25D_11D3_B82B_00105A60F930__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PPMYAddress.h : header file
//

/////////////////////////////////////////////////////////////////
// CPPMyAddress dialog

class CPPMyAddress : public CDialog
{
// Construction
public:
    CPPMyAddress(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CPPMyAddress)
    enum { IDD = IDD_PP_MYADDRESS };
    CComboBox    m_cbCountry;
    CString      m_sCity;
    CString      m_sState;
    CString      m_sAddress;
    CString      m_sZip;
    CString      m_sTelHome;
    CString      m_sTelMobile;
    CString      m_sTelOther;
    CString      m_sTelWork;
    CString      m_sTelFax;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPPMyAddress)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CPPMyAddress)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

```

PPMyAddress.h

```
#endif //  
!defined(AFX_PPMYADDRESS_H__0E8E8442_B25D_11D3_B82B_00105A60F930__INCLUDED_)
```

PPMyInfo.cpp

```
// PPMYInfo.cpp : implementation file
//

#include "stdafx.h"
#include "uc2.h"
#include "PPMyInfo.h"

#include "PPActor.h"
#include "PSJoinChannel.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/PhSprite.h"
#include "ResMan.h"

#include "Behavior.h"    // CActorDesc

#include "BlockSock.h"
#include "ConMan.h"
#include "MemberInfo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;
extern CConMan gConMan;

////////////////////////////////////
// CPPMyInfo property page

IMPLEMENT_DYNCREATE(CPPMyInfo, CPropertyPage)

CPPMyInfo::CPPMyInfo() : CPropertyPage(CPPMyInfo::IDD)
{
    //{AFX_DATA_INIT(CPPMyInfo)
    m_nAge = 0;
    //}AFX_DATA_INIT
}

CPPMyInfo::~CPPMyInfo()
{
}

void CPPMyInfo::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CPPMyInfo)
    DDX_Control(pDX, IDC_LB_ACTORS, m_lbActors);
    DDX_Control(pDX, IDC_COMBO_WORK, m_cbWork);
    DDX_Control(pDX, IDC_COMBO_SEX, m_cbSex);
    DDX_Control(pDX, IDC_COMBO_ETHNIC, m_cbEthnic);
    DDX_Text(pDX, IDC_EDIT_AGE, m_nAge);
    //}AFX_DATA_MAP
}
```



```

    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPMyInfo, CPropertyPage)
    //{AFX_MSG_MAP(CPPMyInfo)
    ON_LBN_SELCHANGE(IDC_LB_ACTORS, OnSelchangeLbActors)
    ON_CBN_SELCHANGE(IDC_COMBO_SEX, OnSelchangeComboSex)
    ON_WM_HSCROLL()
    ON_WM_PAINT()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPPMyInfo message handlers

BOOL CPPMyInfo::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    TRACE0("CPPMyInfo::OnInitDialog()\n");
    CPropertyPage::OnInitDialog();

    m_bChanged = FALSE;
    m_cbSex.SetCurSel(m_nSex);
    m_cbEthnic.SetCurSel(m_nEthnic);
    m_cbWork.SetCurSel(m_nWork);
    // m_cbLanguage.SetCurSel(m_nLanguage);

    for (int i=0; i < gResMan.GetNumActorDescs(); i++)
    {
        CActorDesc* pAD = gResMan.GetActorDesc(i);
        if (pAD)
        {
            CString* pNick = pAD->GetNick();
            if (pNick)
                m_lbActors.AddString(*pNick);
        }
    }
    m_lbActors.SetCurSel(0);

    // m_rcActor.left = 24; // syc 0715
    // m_rcActor.top = 21; // syc 0715
    UpdateActorImage();

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CPPMyInfo::OnSelchangeLbActors()
{
    m_nCharID = m_lbActors.GetCurSel();
    m_nCellID = 0;
    UpdateActorImage();
    // m_bChanged= TRUE;
    SetModified();
}

```

```

}

void CPPMyInfo::OnSelchangeComboSex()
{
    m_nSex = m_cbSex.GetCurSel();
    TRACE("Sex:%d\n", m_nSex);
    m_bChanged= TRUE;
    SetModified();
}

void CPPMyInfo::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_ACTOR);

    if (pScrollBar == pSB)
    {
        if (!m_pPSActor)
            return;
        int nCells = m_pPSActor->GetNumCells();
        int nMin, nMax;
        pSB->GetScrollRange(&nMin, &nMax);
        switch (nSBCode)
        {
            case SB_THUMBPOSITION:
                m_nCellID = nPos;
                pSB->SetScrollPos(nPos);
                break;
            case SB_LINELEFT:
                m_nCellID = pSB->GetScrollPos() - 1;
                if (m_nCellID < nMin)
                    m_nCellID = nMax; // Wrap
                pSB->SetScrollPos(m_nCellID);
                break;
            case SB_LINERIGHT:
                m_nCellID = pSB->GetScrollPos() + 1;
                if (m_nCellID > nMax)
                    m_nCellID = nMin; // Wrap
                pSB->SetScrollPos(m_nCellID);
                break;
            case SB_PAGELEFT:
                m_nCellID = pSB->GetScrollPos() - (nMax - nMin) / 5;
                if (m_nCellID < nMin)
                    m_nCellID = nMin;
                pSB->SetScrollPos(m_nCellID);
                break;
            case SB_PAGERIGHT:
                m_nCellID = pSB->GetScrollPos() + (nMax - nMin) / 5;
                if (m_nCellID > nMax)
                    m_nCellID = nMax;
                pSB->SetScrollPos(m_nCellID);
                break;
        }
        InvalidateRect(&m_rcActor, FALSE);
        UpdateData(FALSE);
    }
    CPropertyPage::OnHScroll(nSBCode, nPos, pScrollBar);
}

```

```

}
```

```

// m_nCharID can be set externally by calling CPSJoinChannel::SetMemberInfo
void CPPMyInfo::UpdateActorImage()
{
```

```
    CActorDesc* pAD = gResMan.GetActorDesc(m_nCharID);
```

```
    if (!pAD)
```

```
    {
```

```
        TRACE0("CPPMyInfo - ActorDesc not found!\n");
```

```
        return;
```

```
    }
```

```
    CString* pstrRes = pAD->GetResName();
```

```
    if (!pstrRes)
```

```
    {
```

```
        TRACE0("CPPMyInfo - ActorDesc - ResName not found!\n");
```

```
        return;
```

```
    }
```

```
    CString strResName(*pstrRes);
```

```
    // We'll not reuse DIB here:
```

```
    // Since this image is not necessary to be resident in memory.
```

```
    // CPhasedSprite* pPS = gResMan.LoadPhasedSprite('A', strResName, FALSE);
```

```
    // (strResName, FALSE); // sync 0501
```

```
    CPhasedSprite* pPS = gResMan.LoadPhasedSprite(strResName, FALSE);
```

```
    if (!pPS)
```

```
    {
```

```
        delete pPS;
```

```
        strResName += " not found!";
```

```
        AfxMessageBox(strResName);
```

```
        return;
```

```
    }
```

```
    if (m_pPSActor)
```

```
        delete m_pPSActor; // Delete previously allocated resource
```

```
    m_pPSActor = pPS;
```

```
    m_nCellID = 0;
```

```
    m_pPSActor->SetCell(m_nCellID);
```

```
    CRect rc;
```

```
    m_pPSActor->GetRect(rc);
```

```
    m_rcActor.right = m_rcActor.left + rc.Width();
```

```
    m_rcActor.bottom = m_rcActor.top + rc.Height();
```

```
    CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_ACTOR);
```

```
    ASSERT(pSB);
```

```
    pSB->SetWindowPos(NULL, m_rcActor.left, m_rcActor.bottom+2,
                        m_rcActor.Width()*2, 15,
```

```
                        SWP_NOZORDER | SWP_NOACTIVATE);
```

```
    pSB->SetScrollRange(0, m_pPSActor->GetNumCells()-1);
```

```
    pSB->SetScrollPos(0);
```

```
    m_lbActors.SetCurSel(m_nCharID);
```

```
    InvalidateRect(&m_rcActor);
```

```

}
```

```

void CPPMyInfo::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    CPSJoinChannel* pPSJC = (CPSJoinChannel*)GetParent();
    ASSERT(pPSJC);
    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (pPSJC->GetPalette())
    {
        pPalOld = dc.SelectPalette(pPSJC->GetPalette(), FALSE); //
bForceBackground = FALSE
        // pDC->RealizePalette(); // we realize in response to
WM_QUERYNEWPALETTE
    }

    if (m_pPSActor)
    {
        m_pPSActor->SetCell(m_nCellID);
        CPoint ptLT(m_rcActor.left, m_rcActor.top);
        m_pPSActor->Draw(&dc, ptLT);
    }

    if (pPalOld)
        dc.SelectPalette(pPalOld, FALSE);
    // Do not call CPropertyPage::OnPaint() for painting messages
}

```

PPMyInfo.h

```
#if !defined(AFX_PP_MYINFO_H__4DDADF81_B0E7_11D3_B82B_00105A60F930__INCLUDED_)
#define AFX_PP_MYINFO_H__4DDADF81_B0E7_11D3_B82B_00105A60F930__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PPMyInfo.h : header file
//

class CPhasedSprite;

//////////////////////////////////////
// CPPMyInfo dialog

class CPPMyInfo : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPMyInfo)

// Construction
public:
    CPPMyInfo();
    ~CPPMyInfo();

    int         m_nCharID;
    int         m_nSex;           // 0: ?, 1: Male, 2: Female
    int         m_nEthnic;
    int         m_nLanguage;
    int         m_nWork;

    BOOL        m_bChanged;

// Dialog Data
    //{AFX_DATA(CPPMyInfo)
    enum { IDD = IDD_PP_MYINFO };
    CListBox     m_lbActors;
    CComboBox    m_cbWork;
    CComboBox    m_cbSex;
    CComboBox    m_cbEthnic;
    CButton      m_btnUploadPhoto;
    UINT         m_nAge;
    //}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{AFX_VIRTUAL(CPPMyInfo)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    void         UpdateActorImage();

    int         m_nCellID;
    CPhasedSprite* m_pPSActor;
};
```

PPMyInfo.h

```
CRect          m_rcActor;

// Generated message map functions
//{{AFX_MSG(CPPMyInfo)
virtual BOOL OnInitDialog();
afx_msg void OnSelchangeLbActors();
afx_msg void OnSelchangeComboSex();
afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar);
afx_msg void OnPaint();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PPMYINFO_H__4DDADF81_B0E7_11D3_B82B_00105A60F930__INCLUDED_)
```

```
// PPOtherInfo.cpp : implementation file
//

#include "stdafx.h"
#include "uc2.h"
#include "PPOtherInfo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CPPOtherInfo property page

IMPLEMENT_DYNCREATE(CPPOtherInfo, CPropertyPage)

CPPOtherInfo::CPPOtherInfo() : CPropertyPage(CPPOtherInfo::IDD)
{
    //{{AFX_DATA_INIT(CPPOtherInfo)
    m_sAge = _T("");
    m_sAlias = _T("");
    m_sEmail = _T("");
    m_sEthnic = _T("");
    m_sFName = _T("");
    m_sHomepage = _T("");
    m_sLanguage = _T("");
    m_sLName = _T("");
    m_sSex = _T("");
    m_sWork = _T("");
    //}}AFX_DATA_INIT
}

CPPOtherInfo::~CPPOtherInfo()
{
}

void CPPOtherInfo::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPOtherInfo)
    DDX_Text(pDX, IDC_ST_AGE, m_sAge);
    DDX_Text(pDX, IDC_ST_ALIAS, m_sAlias);
    DDX_Text(pDX, IDC_ST_EMAIL, m_sEmail);
    DDX_Text(pDX, IDC_ST_ETHNIC, m_sEthnic);
    DDX_Text(pDX, IDC_ST_FNAME, m_sFName);
    DDX_Text(pDX, IDC_ST_HOMEPAGE, m_sHomepage);
    DDX_Text(pDX, IDC_ST_LANGUAGE, m_sLanguage);
    DDX_Text(pDX, IDC_ST_LNAME, m_sLName);
    DDX_Text(pDX, IDC_ST_SEX, m_sSex);
    DDX_Text(pDX, IDC_ST_WORK, m_sWork);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPOtherInfo, CPropertyPage)
```

PPOtherInfo.cpp

```
//{{AFX_MSG_MAP(CPPOtherInfo)
    // NOTE: the ClassWizard will add message map macros here
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPPOtherInfo message handlers
```


PPOtherInfo.h

```

#if
!defined(AFX_PPOTHERINFO_H__0E8E8441_B25D_11D3_B82B_00105A60F930__INCLUDED_)
#define AFX_PPOTHERINFO_H__0E8E8441_B25D_11D3_B82B_00105A60F930__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PPOtherInfo.h : header file
//

/////////////////////////////////////////////////////////////////
// CPPOtherInfo dialog

class CPPOtherInfo : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPOtherInfo)

// Construction
public:
    CPPOtherInfo();
    ~CPPOtherInfo();

// Dialog Data
   //{{AFX_DATA(CPPOtherInfo)
    enum { IDD = IDD_PP_OTHERINFO };
    CString        m_sAge;
    CString        m_sAlias;
    CString        m_sEmail;
    CString        m_sEthnic;
    CString        m_sFName;
    CString        m_sHomepage;
    CString        m_sLanguage;
    CString        m_sLName;
    CString        m_sSex;
    CString        m_sWork;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPPOtherInfo)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CPPOtherInfo)
    // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}

```

PPOtherInfo.h

// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //

!defined(AFX_PPOTHERINFO_H__0E8E8441_B25D_11D3_B82B_00105A60F930__INCLUDED_)

PPPPaymentInfo.cpp

```
// PPPaymentInfo.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "PPPaymentInfo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CPPPaymentInfo property page

IMPLEMENT_DYNCREATE(CPPPaymentInfo, CPropertyPage)

CPPPaymentInfo::CPPPaymentInfo() : CPropertyPage(CPPPaymentInfo::IDD)
{
    //{{AFX_DATA_INIT(CPPPaymentInfo)
    m_strCardNum = _T("");
    m_strExpDate = _T("");
    //}}AFX_DATA_INIT
}

CPPPaymentInfo::~CPPPaymentInfo()
{
}

void CPPPaymentInfo::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPPaymentInfo)
    DDX_Control(pDX, IDC_COMBO1, m_cbCredit);
    DDX_Text(pDX, IDC_EDIT_CARDNUM, m_strCardNum);
    DDX_Text(pDX, IDC_EDIT_EXPDATE, m_strExpDate);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPPaymentInfo, CPropertyPage)
    //{{AFX_MSG_MAP(CPPPaymentInfo)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// CPPPaymentInfo message handlers

BOOL CPPPaymentInfo::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    m_cbCredit.SetCurSel(0);
    m_strExpDate = _T("MM/YY");
    m_strCardNum = _T("XXXX-XXXX-XXXX-XXXX");
    UpdateData(FALSE); // Write
}
```

```
    return TRUE;    // return TRUE unless you set the focus to a control
                    // EXCEPTION: OCX Property Pages should return FALSE
}
```

PPPaymentInfo.h

```

#if
!defined(AFX_PPMPAYMENTINFO_H__19BBF282_474C_11D2_BCFD_0080C7EADFBB__INCLUDED_
)
#define AFX_PPMPAYMENTINFO_H__19BBF282_474C_11D2_BCFD_0080C7EADFBB__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PPPaymentInfo.h : header file
//

////////////////////////////////////
// CPMPPaymentInfo dialog

class CPMPPaymentInfo : public CPropertyPage
{
    DECLARE_DYNCREATE(CPMPPaymentInfo)

// Construction
public:
    CPMPPaymentInfo();
    ~CPMPPaymentInfo();

// Dialog Data
    //{{AFX_DATA(CPMPPaymentInfo)
    enum { IDD = IDD_PP_PAYMENT };
    CComboBox    m_cbCredit;
    CString      m_strCardNum;
    CString      m_strExpDate;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPMPPaymentInfo)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CPMPPaymentInfo)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PPMPAYMENTINFO_H__19BBF282_474C_11D2_BCFD_0080C7EADFBB__INCLUDED_
)
    
```

PPShoppingCart.cpp

```
// PPShoppingCart.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "PPShoppingCart.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CPPShoppingCart property page

IMPLEMENT_DYNCREATE(CPPShoppingCart, CPropertyPage)

CPPShoppingCart::CPPShoppingCart() : CPropertyPage(CPPShoppingCart::IDD)
{
    //{{AFX_DATA_INIT(CPPShoppingCart)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

CPPShoppingCart::~CPPShoppingCart()
{
}

void CPPShoppingCart::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPShoppingCart)
    DDX_Control(pDX, IDC_CART, m_lcCart);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPShoppingCart, CPropertyPage)
    //{{AFX_MSG_MAP(CPPShoppingCart)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPPShoppingCart message handlers

BOOL CPPShoppingCart::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    // Prepare List Control for Channels list
    // m_lcCart.SetImageList(&m_ilChannel, LVSIL_SMALL);

    char* szColumn[] = {"QNTY", "ITEM", "ID#", "PRICE", "TOTAL"};
    int nWidth[] = {40, 130, 40, 70, 70};

    LV_COLUMN lvC; // list view column structure

```

```

        lvC.mask      = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM;    //
valid members
        lvC.fmt       = LVCFMT_LEFT;    // left-align column

        // Add the columns.
        for (int i=0; i < sizeof(nWidth)/sizeof(nWidth[0]); i++)
        {
            lvC.cx      = nWidth[i];    // width of column in
pixels
            lvC.iSubItem = i;
            lvC.pszText  = szColumn[i];
            if (m_lcCart.InsertColumn(i, &lvC) == -1)
                return NULL;
        }

#define NUM_ITEMS 4
        LPTSTR aszQuantity[NUM_ITEMS] = {"1", "2", "1", "3"};
        LPTSTR aszItem[NUM_ITEMS]     = {"SM 518", "SM 520", "VTR", "SM
525"};
        LPTSTR aszID[NUM_ITEMS]       = {"1353", "1742", "6727", "5917"};
        LPTSTR aszPrice[NUM_ITEMS]    = {"$18,000", "$22,000", "$330",
"$28,000"};
        LPTSTR aszTotal[NUM_ITEMS]    = {"$18,000", "$56,000", "$56,330",
"$92,000"};
        LV_ITEM          lvI; // list view item structure
        for (i=0; i < NUM_ITEMS; i++)
        {
            ::ZeroMemory(&lvI, sizeof(lvI));
            lvI.mask      = LVIF_TEXT | LVIF_IMAGE | LVIF_PARAM |
LVIF_STATE;
            lvI.state     = 0;
            lvI.stateMask = 0;

            lvI.iItem     = i;
            lvI.iSubItem  = 0;
            // The parent window is responsible for storing the text.
            // The list view control will send an LVN_GETDISPINFO
            // when it needs the text to display.
            lvI.pszText   = aszQuantity[i];
            lvI.cchTextMax = MIC_MAX_CHANNEL_NAME_LENGTH_MIC; // 63
            //
            lvI.iImage    = iImage;
            //
            lvI.lParam    = (LPARAM)picsProp;
            m_lcCart.InsertItem(&lvI);
            m_lcCart.SetItemText(i, 1, aszItem[i]);
            m_lcCart.SetItemText(i, 2, aszID[i]);
            m_lcCart.SetItemText(i, 3, aszPrice[i]);
            m_lcCart.SetItemText(i, 4, aszTotal[i]);
        }
#undef NUM_ITEMS

        return TRUE; // return TRUE unless you set the focus to a control
                    // EXCEPTION: OCX Property Pages should return FALSE
    }

```

PPShoppingCart.h

```

#if
!defined(AFX_PPSHOPPINGCART_H__19BBF281_474C_11D2_BCFD_0080C7EADFBB__INCLUDED
_)
#define AFX_PPSHOPPINGCART_H__19BBF281_474C_11D2_BCFD_0080C7EADFBB__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// PPSHoppingCart.h : header file
//

////////////////////////////////////
// CPPShoppingCart dialog

class CPPShoppingCart : public CPropertyPage
{
    DECLARE_DYNCREATE(CPPShoppingCart)

// Construction
public:
    CPPShoppingCart();
    ~CPPShoppingCart();

// Dialog Data
   //{{AFX_DATA(CPPShoppingCart)
    enum { IDD = IDD_PP_CART };
    CListCtrl    m_lcCart;
    //}}AFX_DATA

// Overrides
    // ClassWizard generate virtual function overrides
   //{{AFX_VIRTUAL(CPPShoppingCart)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(CPPShoppingCart)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PPSHOPPINGCART_H__19BBF281_474C_11D2_BCFD_0080C7EADFBB__INCLUDED
_)

```



```
//
=====
// File: P R O G R E S S . C P P
//
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
// PARTICULAR PURPOSE.
//
// Description:
//
// This sample demonstrates using a URL moniker to download information.
// The key routines include the implementation of IBindStatusCallback
// and the CDownload::DoDownload routine, which creates and binds to the
// URL moniker.
//
// Instructions:
//
// To use this sample:
// * build it using the NMAKE command. NMAKE will create PROGRESS.EXE.
// * run PROGRESS.EXE. specify the resource to download by passing an
//   URL on the command-line. use no command-line argument to default to
//   downloading "http://www.msn.com".
// * The program displays a dialog box containing information about the
//   download:
//   - a status message, describing the current status of the download
//   - a progress message, describing the amount of information that
//     has been downloaded.
//   - a text box, which displays chunks of the download information as
//     it arrives.
// * Press the "GO" button to begin the download.
//
// Sample update:-
// * New feature include Progress Bar to indicate progress of the download
// * Edit box replaces the old text box. Also the whole file can be
//   viewed.
//   If the file exceeds 32KB then only the first 32 KB from the last data
//   pull will be displayed. If the pull exceeds 32 KB, then only the
//   first
//   32 KB of the last Read will be displayed.
//
//
//
// File updated by Jason Strayer 27-Aug-1997
// File updated by Jobi George 19-June-1996
// File updated by Ramesha Gopalakrishna 28-June-1996
// File updated by Oliver Wallace 9-July-1996
// File updated by Soomin Kim 6-April-1998
// Copyright 1995-1997 Microsoft Corporation. All Rights Reserved.
//
=====
#include "stdafx.h"
#include "urlmon.h"
#include "wininet.h"
#include "resource.h"
#include "Prog.h"
```

```

//
=====
//                                     CBindStatusCallback Implementation
//
=====

// -----
-
// %%Function: CBindStatusCallback::CBindStatusCallback
// -----
-
CBindStatusCallback::CBindStatusCallback(HWND hwndFrame, LPCTSTR szFile)
{
    TRACE("CBindStatusCallback::CBindStatusCallback\n");
    m_hFrame      = hwndFrame;
    m_pbinding     = NULL;
    m_pstm         = NULL;
    m_cRef         = 1;
    m_pFile        = NULL;
    m_strFile      = szFile;
} // CBindStatusCallback

// -----
-
// %%Function: CBindStatusCallback::~CBindStatusCallback
// -----
-
CBindStatusCallback::~CBindStatusCallback()
{
    TRACE("CBindStatusCallback::~CBindStatusCallback\n");
} // -CBindStatusCallback

/*
inline void CBindStatusCallback::SetWndText(HWND hwnd, LPCWSTR szText)
{
    if (IsWindow(hwnd))
    {
        char rgchBuf[INTERNET_MAX_PATH_LENGTH];
        WideCharToMultiByte(CP_ACP, 0, szText, -1, rgchBuf,
INTERNET_MAX_PATH_LENGTH, 0, 0);
        SetWindowText(hwnd, rgchBuf);
    }
}
*/

// -----
-
// %%Function: CBindStatusCallback::QueryInterface
// -----
-
STDMETHODIMP CBindStatusCallback::QueryInterface(REFIID riid, void** ppv)
{
    *ppv = NULL;

    if (riid==IID_IUnknown || riid==IID_IBindStatusCallback)
    {
        *ppv = this;
    }
}

```

```

        AddRef();
        return S_OK;
    }
    return E_NOINTERFACE;
} // CBindStatusCallback::QueryInterface

// -----
// %%Function: CBindStatusCallback::OnStartBinding
// -----
-
STDMETHODIMP CBindStatusCallback::OnStartBinding(DWORD dwReserved, IBinding*
pbinding)
{
    TRACE("CBindStatusCallback::OnStartBinding\n");
    if (m_pbinding)
        m_pbinding->Release();
    m_pbinding = pbinding;
    if (m_pbinding)
    {
        m_pbinding->AddRef();
        PostMessage(CMD_BIND_START); // SetStatus(L"Status: Starting to
bind...");
    }
    TRACE0("m_pFile = new CFile;\n");
    m_pFile = new CFile;
    if (!m_pFile->Open(m_strFile, CFile::modeReadWrite | CFile::modeCreate
| CFile::shareExclusive))
    {
        delete m_pFile;
        m_pFile = NULL;
        AfxMessageBox("Failed to open file");
        return FALSE;
    }
    PostMessage(CMD_BIND_FILE_OPENED);
    return S_OK;
} // CBindStatusCallback::OnStartBinding

// -----
// %%Function: CBindStatusCallback::OnStopBinding
// -----
-
STDMETHODIMP CBindStatusCallback::OnStopBinding(HRESULT hrStatus, LPCWSTR
pszError)
{
    TRACE("CBindStatusCallback::OnStopBinding\n");
    if (hrStatus)
        PostMessage(CMD_BIND_FAILED); // SetStatus(L"Status: File
download Failed.");

    if (m_pbinding)
    {
        m_pbinding->Release();
        m_pbinding = NULL;
    }
}

```

```

        CloseFile();
        return S_OK;
    } // CBindStatusCallback::OnStopBinding

void CBindStatusCallback::CloseFile()
{
    if (m_pFile)
    {
        TRACE("m_pFile->Close();\n");
        m_pFile->Close();
        delete m_pFile;
        m_pFile = NULL;
        PostMessage(CMD_BIND_FILE_CLOSED);
    }
}

// -----
// %%Function: CBindStatusCallback::GetPriority
// -----
STDMETHODIMP CBindStatusCallback::GetPriority(LONG* pnPriority)
{
    return E_NOTIMPL;
} // CBindStatusCallback::GetPriority

// -----
// %%Function: CBindStatusCallback::OnLowResource
// -----
STDMETHODIMP CBindStatusCallback::OnLowResource(DWORD dwReserved)
{
    return E_NOTIMPL;
} // CBindStatusCallback::OnLowResource

// -----
// %%Function: CBindStatusCallback::OnProgress
// -----
STDMETHODIMP CBindStatusCallback::OnProgress(ULONG ulProgress, ULONG
ulProgressMax,
                                                    ULONG
ulStatusCode, LPCWSTR szStatusText)
{
    TRACE("CBindStatusCallback::OnProgress(%lu)\n", ulProgress);
    char sz[255];

    if (szStatusText)
        WideCharToMultiByte(CP_ACP, 0, szStatusText, -1, sz, 255, 0, 0);

    if (lstrlen(sz) > 0)
    {
        char msg[256];
        char buf[256];
        // WCHAR out[256*2];
    }
}

```

```

        char* p = strrchr(sz, '/');
        if (p)
        {
            p++;
            strcpy(buf, p);
            wsprintf(msg, "%s %d of %d", buf, ulProgress, (ulProgress >
ulProgressMax) ? ulProgress : ulProgressMax);
            // MultiByteToWideChar(CP_ACP, 0, msg, -1, out, sizeof(out));

            SendMessage(CMD_BIND_PROGRESS_MSG, 0, (LPARAM)msg);
        }
        TRACE("%lu, %lu\n", ulProgress, ulProgressMax);
        SendMessage(CMD_BIND_PROGRESS_BAR, (WPARAM)ulProgress,
(LPARAM)ulProgressMax); // by value
    }
    return NOERROR;
} // CBindStatusCallback::OnProgress

// -----
// %%Function: CBindStatusCallback::GetBindInfo
// -----
-
STDMETHODIMP CBindStatusCallback::GetBindInfo(DWORD* pgrfBINDF, BINDINFO*
pbindInfo)
{
    if (!pbindInfo || !pbindInfo->cbSize || !pgrfBINDF)
        return E_INVALIDARG;

    *pgrfBINDF = BINDF_ASYNCHRONOUS | BINDF_ASYNCSTORAGE | BINDF_PULLDATA |
        BINDF_GETNEWESTVERSION | BINDF_NOWRITECACHE;

    // remember incoming cbSize
    ULONG cbSize = pbindInfo->cbSize;
    // zero out structure
    ::ZeroMemory(pbindInfo, cbSize);

    // restore cbSize
    pbindInfo->cbSize = cbSize;
    pbindInfo->dwBindVerb = BINDVERB_GET;

    return S_OK;
} // CBindStatusCallback::GetBindInfo

// -----
// %%Function: CBindStatusCallback::OnDataAvailable
// -----
-
STDMETHODIMP CBindStatusCallback::OnDataAvailable(DWORD grfBSCF, DWORD
dwSize, FORMATETC* pfmtetc, STGMEDIUM* pstgmed)
{
    TRACE("CBindStatusCallback::OnDataAvailable(%ld)\n", dwSize);
    HRESULT hr=S_OK;

    // Get the Stream passed

```

```

if (BSCF_FIRSTDATANOTIFICATION & grfBSCF)
{
    if (!m_pstm && (pstgmed->tymed == TYMED_ISTREAM))
    {
        m_pstm = pstgmed->pstm;
        if (m_pstm)
            m_pstm->AddRef();
    }
}

// If there is some data to be read then go ahead and read them
if (m_pstm && (dwSize > 0))
{
    DWORD dwRead = dwSize;          // Minimum amount available that
hasn't been read
    DWORD dwActuallyRead = 0;       // Placeholder for amount read
during this pull

    if (dwRead > 0)
    {
        do
        {
            char* pNewstr = new char[dwRead + 1];
            if (!pNewstr)
                return S_FALSE;
            hr = m_pstm->Read(pNewstr, dwRead, &dwActuallyRead);
            pNewstr[dwActuallyRead] = 0;
            // If we really read something then lets add it to
the Edit box

            if (m_pFile && (dwActuallyRead > 0))
            {
                TRY
                {
                    m_pFile->Write(pNewstr, dwActuallyRead);
                }
                CATCH(CFileException, e)
                {
                    TRACE0("Failed to write file.\n");
                    return FALSE;
                }
            } END_CATCH
            delete [] pNewstr;
        } while (!(hr == E_PENDING || hr == S_FALSE) &&
SUCCEEDED(hr));
    }
    // if (m_pstm && dwSize > 0)

    if (BSCF_LASTDATANOTIFICATION & grfBSCF)
    {
        if (m_pstm)
            m_pstm->Release();
        hr = S_OK; // If it was the last data then we should return S_OK
as we just finished reading everything
        PostMessage(CMD_BIND_DOWNLOAD_DONE);          // SetStatus(L"Status:
File downloaded.");
        TRACE0("Status: File downloaded.\n");
    }
}

```

```

        return hr;
    } // CBindStatusCallback::OnDataAvailable

// -----
// %%Function: CBindStatusCallback::OnObjectAvailable
// -----
-
STDMETHODIMP CBindStatusCallback::OnObjectAvailable(REFIID riid, IUnknown*
punk)
{
    return E_NOTIMPL;
} // CBindStatusCallback::OnObjectAvailable

//
=====
//                                     CDownload Implementation
//
=====

// -----
// %%Function: CDownload::CDownload
// -----
-
CDownload::CDownload(LPCTSTR szURL)
{
    TRACE("CDownload::CDownload\n");
    static WCHAR      szwURL[MAX_PATH]; //
    L"http://www.msn.com";
    MultiByteToWideChar(CP_ACP, 0, szURL, -1, szwURL, MAX_PATH);
    m_url = szwURL;
    m_pmk = 0;
    m_pbc = 0;
    m_pbsc = 0;
} // CDownload

// -----
// %%Function: CDownload::~CDownload
// -----
-
CDownload::~CDownload()
{
    TRACE("CDownload::~CDownload\n");
    if (m_pmk)
        m_pmk->Release();
    if (m_pbc)
        m_pbc->Release();
    if (m_pbsc)
    {
        m_pbsc->Release();
        delete m_pbsc;
    }
} // ~CDownload

```

```

// -----
// %%Function: CDownload::DoDownload
// -----
-
HRESULT CDownload::DoDownload(HWND hwndFrame, LPCTSTR szFile)
{
    TRACE("CDownload::DoDownload\n");
    IStream* pstm = NULL;

    HRESULT hr = CreateURLMoniker(NULL, m_url, &m_pmk);
    if (FAILED(hr))
        goto LErrExit;

    m_pbsc = new CBindStatusCallback(hwndFrame, szFile);
    if (!m_pbsc)
    {
        hr = E_OUTOFMEMORY;
        goto LErrExit;
    }

    hr = CreateBindCtx(0, &m_pbc);
    if (FAILED(hr))
        goto LErrExit;

    hr = RegisterBindStatusCallback(m_pbc, m_pbsc, 0, 0L);
    if (FAILED(hr))
        goto LErrExit;

    hr = m_pmk->BindToStorage(m_pbc, 0, IID_IStream, (void**)&pstm);
    if (FAILED(hr))
        goto LErrExit;

    return hr;

LErrExit:
    if (m_pbc)
    {
        m_pbc->Release();
        m_pbc = NULL;
    }
    if (m_pbsc)
    {
        m_pbsc->Release();
        m_pbsc = NULL;
    }
    if (m_pmk)
    {
        m_pmk->Release();
        m_pmk = NULL;
    }
    if (pstm)
    {
        pstm->Release();
        pstm = NULL;
    }
    return hr;
}

```


Prog.cpp

```
}      // CDownload::DoDownload

void CDownload::CancelDownload()
{
    if (m_pbsc)
    {
        m_pbsc->CloseFile();
    }
}
```

```
//=====
//      (C) Programmed by Kim, Soomin, Apr, 1996
//      Information Technology Institue
//      UNICHAT.COM
//      Original source from Microsoft
//=====

#ifndef __PROG_H
#define __PROG_H

#include "urlmon.h"
#include "wininet.h"

#define BASE      (WM_USER + 200 + 100)
const UINT  CMD_BIND_START          = BASE;
const UINT  CMD_BIND_FAILED         = BASE+1;
const UINT  CMD_BIND_FILE_OPENED    = BASE+2;
const UINT  CMD_BIND_FILE_CLOSED    = BASE+3;
const UINT  CMD_BIND_PROGRESS_MSG   = BASE+4; //
SendMessage(CMD_BIND_PROGRESS_MSG, 0, (LPARAM)msg);
const UINT  CMD_BIND_PROGRESS_BAR   = BASE+5; //
SendMessage(CMD_BIND_PROGRESS_BAR, (WPARAM)ulProgress,
(LPARAM)ulProgressMax); // by value
const UINT  CMD_BIND_DOWNLOAD_DONE  = BASE+6;
#undef BASE

// %%Classes: -----
-
class CBindStatusCallback : public IBindStatusCallback
{
public:
    // IUnknown methods
    STDMETHODCALLTYPE QueryInterface(REFIID riid, void ** ppv);
    STDMETHODCALLTYPE AddRef()      { return m_cRef++; }
    STDMETHODCALLTYPE Release()     { return m_cRef--; }
    //if (--m_cRef == 0) { delete this; return 0; } return m_cRef; }

    // IBindStatusCallback methods
    STDMETHODCALLTYPE OnStartBinding(DWORD dwReserved, IBinding* pbinding);
    STDMETHODCALLTYPE GetPriority(LONG* pnPriority);
    STDMETHODCALLTYPE OnLowResource(DWORD dwReserved);
    STDMETHODCALLTYPE OnProgress(ULONG ulProgress, ULONG ulProgressMax, ULONG
ulStatusCode,
                                LPCWSTR pwzStatusText);
    STDMETHODCALLTYPE OnStopBinding(HRESULT hrResult, LPCWSTR szError);
    STDMETHODCALLTYPE GetBindInfo(DWORD* pgrfBINDF, BINDINFO* pbindinfo);
    STDMETHODCALLTYPE OnDataAvailable(DWORD grfBSCF, DWORD dwSize, FORMATETC
*pfmtetc,
                                STGMEDIUM* pstgmed);
    STDMETHODCALLTYPE OnObjectAvailable(REFIID riid, IUnknown* punk);

    // constructors/destructors
    CBindStatusCallback(HWND hwndFrame, LPCTSTR szFile);
    ~CBindStatusCallback();

    BOOL PostMessage(UINT message, WPARAM wParam=0L, LPARAM lParam=0L)
const
```

Prog.h

```
        { return ::PostMessage(m_hFrame, message, wParam, lParam);
    }
    BOOL SendMessage(UINT message, WPARAM wParam=0L, LPARAM lParam=0L)
const
        { return ::SendMessage(m_hFrame, message, wParam, lParam);
    }

    void CloseFile();

    // data members
    DWORD      m_cRef;
    IBinding*   m_pbinding;
    IStream*    m_pstm;
    HWND        m_hFrame;
    CFile*      m_pFile;
    CString     m_strFile;
};

class CDownload
{
public:
    CDownload(LPCTSTR szURL);
    ~CDownload();
    HRESULT DoDownload(HWND hwndFrame, LPCTSTR szFile);
    void CancelDownload();
    LPCWSTR m_url;

private:
    IMoniker* m_pmk;
    IBindCtx* m_pbc;
    CBindStatusCallback* m_pbsc;
};

#endif // __PROG_H
```

ProgressDlg.cpp

```
// ProgressDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "ProgressDlg.h"
#include "Prog.h"
#include "ResMan.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"
#include "MainFrm.h"
#include "LoginDlg.h"    // USERDEF_HOST
#include "Parser.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CParser gParser;
extern CResMan gResMan;

const CRect RECT_EDIT(14, 14, 407, 227);
const CRect RECT_PROGRESS(82, 233, 342, 250);
const CRect RECT_PROGRESS_BAR(77, 253, 342, 262);
const CRect RECT_DISPLAY(82, 270, 338, 287);
LPCTSTR URLSERVER      = _T("ftp://");
LPCTSTR U2SERVERPATH   = _T("/UniChat/");
LPCTSTR U2MOD_TXT      = _T("u2mod.txt");           // Message of the Day
#ifdef _KOREAN
LPCTSTR TOKEN_MOD      = _T("MOD");
#else
LPCTSTR TOKEN_MOD      = _T("MODE");
#endif
LPCTSTR TOKEN_FILELIST = _T("FILELIST");

////////////////////////////////////
// CProgressDlg dialog

CProgressDlg::CProgressDlg(CWnd* pParent /*=NULL*/)
: CDialog(CProgressDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CProgressDlg)
    m_strDisplay = _T("");
    m_strProgress = _T("");
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in
    Win32
    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    ASSERT(pMF);
    m_NullBrush.CreateStockObject(NULL_BRUSH);
    m_ftMessage.CreateFont(-12, 0, 0, 0, FW_NORMAL, // FW_BOLD,
        FALSE, FALSE, 0, // bItalic, bUnderline, cStrikeOut
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH || FF_DONTCARE,
```

ProgressDlg.cpp

```

#ifdef _KOREAN
        "UAAA");
#else
        "Arial");
#endif

    m_pPal = NULL;    // Set it NULL before loading DIB

    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + "U2Login|LoginBk.bmp");
    m_pDIBBack = new CDIB;
    if (!m_pDIBBack->Load(strFile))
    {
        delete m_pDIBBack;
        m_pDIBBack = NULL;
        return;
    }

    if (pMF->Is256Palette())
    {
        // Use mainframe's palette to avoid color flickering
        m_pPal = pMF->GetPalette();
        m_pDIBBack->MapColorsToPalette(m_pPal);
        m_bPaletteCreated = FALSE;
    }
    else // Use original palette in the file for TRUE color system
    {
        // Create the palette from the DIB.
        CDIBPal* pDIBPal;
        pDIBPal = new CDIBPal;
        ASSERT(pDIBPal);
        if (!pDIBPal->Create(m_pDIBBack))
        {
            AfxMessageBox("Failed to create palette from DIB file");
            delete pDIBPal;
        }
        m_pPal = pDIBPal; // type casting to parent class
        m_bPaletteCreated = TRUE;
    }

    strFile = strPath + "U2Login|BtnOK.bmp";
    m_btnOK.Load(strFile);
    m_btnOK.SetPalette(m_pPal);

    strFile = strPath + "U2Login|BtnNo.bmp";
    m_btnCancel.Load(strFile);
    m_btnCancel.SetPalette(m_pPal);

    m_pDownload = NULL;
    m_aDI = NULL;
    m_nDis      = 0;

    m_bFirst    = TRUE;
    m_bMOD      = FALSE;
    m_nCurDI    = 0;
    m_strDownloadingFile = strPath + "down.tmp";
    m_bRITModified = FALSE;
}

```

ProgressDlg.cpp

```

CProgressDlg::~CProgressDlg()
{
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal && m_bPaletteCreated)
        delete m_pPal;

    if (m_pDownload)
        delete m_pDownload;
    if (m_aDI)
        delete [] m_aDI;

    // CoUninitialize();
}

void CProgressDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CProgressDlg)
    DDX_Control(pDX, IDC_EDIT_MSG, m_ecMsg);
    DDX_Control(pDX, IDC_PROGRESSBAR, m_pbProgress);
    DDX_Text(pDX, IDC_DISPLAY, m_strDisplay);
    DDX_Text(pDX, IDC_PROGRESS, m_strProgress);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CProgressDlg, CDialog)
    //{{AFX_MSG_MAP(CProgressDlg)
    ON_WM_QUERYNEWPALETTE()
    ON_WM_PALETTECHANGED()
    ON_WM_ERASEBKGD()
    ON_WM_CTLCOLOR()
    ON_WM_SIZE()
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_NCHITTEST()
    //}}AFX_MSG_MAP
    ON_MESSAGE(CMD_BIND_START, OnBindStart)
    ON_MESSAGE(CMD_BIND_FAILED, OnBindFailed)
    ON_MESSAGE(CMD_BIND_FILE_OPENED, OnBindFileOpened)
    ON_MESSAGE(CMD_BIND_FILE_CLOSED, OnBindFileClosed)
    ON_MESSAGE(CMD_BIND_PROGRESS_MSG, OnBindProgressMsg) //
    SendMessage(CMD_BIND_PROGRESS_MSG, 0, (LPARAM)msg);
    ON_MESSAGE(CMD_BIND_PROGRESS_BAR, OnBindProgressBar) //
    SendMessage(CMD_BIND_PROGRESS_BAR, (LPARAM)ulProgress,
        (LPARAM)ulProgressMax);
    ON_MESSAGE(CMD_BIND_DOWNLOAD_DONE, OnBindDownloadDone)
END_MESSAGE_MAP()

////////////////////////////////////
// CProgressDlg message handlers

BOOL CProgressDlg::OnInitDialog()
{
    TRACE0("CProgressDlg::OnInitDialog()\n");
    if (!m_pDIBBack)

```

```

        return FALSE;
    CDialog::OnInitDialog();

    m_btnOK.SubclassDlgItem(IDOK, this);
    m_btnCancel.SubclassDlgItem(IDCANCEL, this);

    CPoint ptLT(349, 238);
    m_btnOK.MoveResize(ptLT);
    ptLT.x = 17;
    m_btnCancel.MoveResize(ptLT);

    InitControl(IDC_EDIT_MSG,          RECT_EDIT);
    InitControl(IDC_PROGRESS,          RECT_PROGRESS);
    InitControl(IDC_PROGRESSBAR,       RECT_PROGRESS_BAR);
    InitControl(IDC_DISPLAY,           RECT_DISPLAY);

    CUC2App* pApp = (CUC2App*)AfxGetApp();
    ASSERT(pApp);
    int nConnType = pApp->RegGetConnType();
    if (nConnType == USERDEF_HOST)
    {
        pApp->RegGetServer(m_strHost);
    }
    else
    {
        CString* pS = gResMan.GetServerIP(nConnType);
        m_strHost = pS ? *pS : _T("88.1.26.2");
    }

    // HRESULT hr = CoInitialize(NULL);
    // if (FAILED(hr))
    //     return FALSE;

    return TRUE; // return TRUE unless you set the focus to a control
}

void CProgressDlg::DownloadFilesList()
{
    CString strU2MOD(URLSERVER + m_strHost + U2SERVERPATH + U2MOD_TXT);
    if (m_pDownload)
        delete m_pDownload;
    m_pDownload = new CDownload(strU2MOD);

    GetDlgItem(IDOK)->EnableWindow(FALSE);
    GetDlgItem(IDCANCEL)->EnableWindow(FALSE);

    // char rgchBuf[INTERNET_MAX_PATH_LENGTH];
    // WideCharToMultiByte(CP_ACP, 0, m_pDownload->m_url, -1, rgchBuf,
    MAX_PATH, 0, 0);
    // SetWindowText(rgchBuf);

    m_strDisplay.LoadString(IDS_PROGRESS_INIT_BIND);
    UpdateDisplay();
    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + U2MOD_TXT);
    m_bMOD = TRUE; // Set MOD mode
    m_pDownload->DoDownload(GetSafeHwnd(), strFile);
}

```

ProgressDlg.cpp

```

        GetDlgItem(IDCANCEL) ->EnableWindow(TRUE);
    }

void CProgressDlg::OnOK()
{
    CDialog::OnOK();
}

void CProgressDlg::OnCancel()
{
    if (m_pDownload)
        m_pDownload->CancelDownload();
    // EndDialog(hwndDlg,0);
    CDialog::OnCancel();
}

LRESULT CProgressDlg::OnBindStart(WPARAM, LPARAM)
{
    m_strDisplay.LoadString(IDS_PROGRESS_PREPARE);
    UpdateDisplay();
    return 0;
}

LRESULT CProgressDlg::OnBindFailed(WPARAM, LPARAM)
{
    m_strDisplay.LoadString(IDS_PROGRESS_DOWNLOAD_FAIL);
    UpdateDisplay();
    return 0;
}

LRESULT CProgressDlg::OnBindFileOpened(WPARAM, LPARAM)
{
    return 0;
}

LRESULT CProgressDlg::OnBindFileClosed(WPARAM, LPARAM)
{
    return 0;
}

LRESULT CProgressDlg::OnBindProgressMsg(WPARAM wParam, LPARAM lParam)
{
    char* szMsg = (char*)lParam;
    if (szMsg && strlen(szMsg))
    {
        m_strProgress = szMsg;
        UpdateProgress();
    }
    return 0;
}

LRESULT CProgressDlg::OnBindProgressBar(WPARAM wParam, LPARAM lParam)
{
    ULONG cProgress      = (ULONG)wParam;
    ULONG maxProgress = (ULONG)lParam;
    m_pbProgress.SetRange(0, 100);
    // m_pbProgress.SetPos(maxProgress ? cProgress * 100 / maxProgress : 0);
}

```



```

        if (m_nDis)
            m_pbProgress.SetPos((m_nCurDI + 1) * 100 / m_nDis);
        return 0;
    }

LRESULT CProgressDlg::OnBindDownloadDone(WPARAM, LPARAM)
{
    m_strDisplay.LoadString(IDS_PROGRESS_DOWNLOADED);
    UpdateDisplay();

    CString strPath(*gResMan.GetResPath());
    CString strFile;

    if (!m_bMOD)        // Normal data files
    {
        if (m_aDI[m_nCurDI].m_bNew)
        {
            strFile = strPath + m_aDI[m_nCurDI].m_strFile;
            CFileStatus fs;
            if (CFile::GetStatus(m_strDownloadingFile, fs))
            {
                if (fs.m_size == m_aDI[m_nCurDI].m_lSize)
                {
                    ::CopyFile(m_strDownloadingFile, strFile,
FALSE);

                    if (CFile::GetStatus(strFile, fs))
                    {
                        fs.m_mtime = m_aDI[m_nCurDI].m_mtime;
                        CFile::SetStatus(strFile, fs);
                        if (strFile.Find(".rit") > 0) // RIT file
                            m_bRITModified = TRUE;
                    }
                    else
                    {
                        strFile += " not found!";
                        AfxMessageBox(strFile);
                    }
                }
            }
        }
        m_nCurDI++;
        DownloadNewFile();
        return 0;
    }

    // Message of the Day
    strFile = strPath + U2MOD_TXT;
    m_bMOD = FALSE;
    CTextFileBuffer tfb(gParser.GetMaxBuffer());
    if (!tfb.Load(strFile))
    {
        strFile += ": Load Error!";
        AfxMessageBox(strFile);
        return -1;
    }
}

```

```

m_nCurDI = 0;
m_nDIs = 0;
while (tfb.ReadString())
{
    gParser.CopyBuffer(tfb.GetString());
    if (gParser.IsCommentLine()) // At first, check if it's a
comment line
        continue;
    CString strBuf;
    if (!gParser.SetLeftToken('#') ||
        !gParser.GetValueRightToken(strBuf, '='))
        continue; // get next line
    if (lstrcmpi(strBuf, "FILELIST") == 0) // matching!
    {
        while (tfb.ReadString())
        {
            gParser.CopyBuffer(tfb.GetString());
            if (gParser.IsCommentLine()) // At first,
check if it's a comment line
                continue;
            if (gParser.SetLeftToken('{') // Begin
            {
                if (gParser.SetLeftToken('}')) // {}
                    break;
                continue; // get next line
            }
            if (gParser.SetLeftToken('}') // End of
contents
                break; // out of while loop
            if (gParser.SetLeftToken('='))
                m_nDIs++; // Anyway, we should increase
the counter
        }
    }
    tfb.SeekToBegin();

    if (m_nDIs)
        m_aDI = new CDownInfo[m_nDIs];

    int di=0;
    CString strTemp;

    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (!gParser.SetLeftToken('#'))
            continue;
        CString strBuf;
        if (!gParser.GetValueRightToken(strBuf, '='))
            continue;

        //////////////////////////////////////
        if (lstrcmpi(strBuf, TOKEN_FILELIST) == 0)
        {

```

```

while (tfb.ReadString())
{
    gParser.CopyBuffer(tfb.GetString());
    if (gParser.IsCommentLine())
        continue;
    if (gParser.SetLeftToken('{'))
    {
        if (gParser.SetLeftToken('}'))
            break;
        continue;
    }
    if (gParser.SetLeftToken('}'))
        break;
    if (!gParser.GetValueRightToken(strTemp, '='))
        continue;
    strTemp.MakeLower();
    m_aDI[di].m_strFile = strTemp;
    gParser.GetValueRightToken(m_aDI[di].m_nType, ',');

    gParser.GetValueRightToken(m_aDI[di].m_lSize, ',');

    int yy, mm, dd, hh, mn;
    gParser.GetValueRightToken(yy, ',');
    gParser.GetValueRightToken(mm, ',');
    gParser.GetValueRightToken(dd, ',');
    gParser.GetValueRightToken(hh, ',');
    gParser.GetValueRightToken(mn, ',');
    if (di >= m_nDIs)
        break;
    CTime t(yy, mm, dd, hh, mn, 0);
    m_aDI[di++].m_mtime = t;
}
}
else if (lstrcmpi(strBuf, TOKEN_MOD) == 0) // matching!
{
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;

        strTemp = tfb.GetString();
        strTemp += _T("\r\n");
        int len = m_ecMsg.GetWindowTextLength();
        m_ecMsg.SetSel(len, len); // starting
        m_ecMsg.ReplaceSel(strTemp);
    }
}

```

```

    }
    DownloadNewFile();

#ifdef _DEBUG
    for (int i=0; i < m_nDIs; i++)
    {
        CTime& t = m_aDI[i].m_mtime;
        TRACE("%s=%d,%ld,%d,%d,%d,%d,%d\n",
            m_aDI[i].m_strFile, m_aDI[i].m_nType, m_aDI[i].m_lSize,
            t.GetYear(), t.GetMonth(), t.GetDay(), t.GetHour(),
            t.GetMinute(), t.GetSecond());
    }
#endif
    return 0;
}

void CProgressDlg::DownloadNewFile()
{
    if (m_nCurDI >= m_nDIs)
    {
        m_strDisplay.LoadString(IDS_PROGRESS_ALL_FILES_DONE);
        UpdateDisplay();
        m_pbProgress.SetPos(100);
        CWnd* pW = GetDlgItem(IDOK);
        pW->EnableWindow(TRUE);
        pW->SetFocus();
        return;
    }
    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + m_aDI[m_nCurDI].m_strFile);
    CTime& tHost = m_aDI[m_nCurDI].m_mtime;
    CFileStatus fsLocal;
    BOOL bFound = CFile::GetStatus(strFile, fsLocal);
    // Compare with local files
    if (!bFound || (fsLocal.m_mtime < tHost))
    {
        CString strDFile(URLSERVER + m_strHost + U2SERVERPATH +
            m_aDI[m_nCurDI].m_strFile);
        if (m_pDownload)
            delete m_pDownload;
        m_pDownload = new CDownload(strDFile);    // Server

        GetDlgItem(IDOK)->EnableWindow(FALSE);
        GetDlgItem(IDCANCEL)->EnableWindow(FALSE);

        m_strDisplay.LoadString(IDS_PROGRESS_INIT_BIND);
        UpdateDisplay();
        m_pDownload->DoDownload(GetSafeHwnd(), m_strDownloadingFile);
        //strFile); // Local
        GetDlgItem(IDCANCEL)->EnableWindow(TRUE);
        m_aDI[m_nCurDI].m_bNew = TRUE;
    }
    else
    {
        m_aDI[m_nCurDI].m_bNew = FALSE;
        PostMessage(CMD_BIND_DOWNLOAD_DONE, 0, 0);    // to proceed to
the next file
    }
}

```

```

    }
}

void CProgressDlg::InitControl(const int nCtrlID, const CRect& rcCtrl)
{
    CWnd* pW = GetDlgItem(nCtrlID);
    ASSERT(pW);
    pW->SetWindowPos(NULL, rcCtrl.left, rcCtrl.top,
                     rcCtrl.Width(), rcCtrl.Height(),
                     SWP_NOZORDER | SWP_NOACTIVATE);
    pW->SetFont(&m_ftMessage, FALSE);    // do not Redraw
}

void CProgressDlg::OnPaletteChanged(CWnd* pFocusWnd)
{
    CDialog::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CProgressDlg::OnQueryNewPalette()
{
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE);    //
foreground
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
        // if (u)
        // {
        //     // Some colors changed so we need to do a repaint.
        //     Invalidate(); // Repaint the lot.
        //     return TRUE; // Say we did something.
        // }
        return FALSE; // Say we did nothing.
    }
}

void CProgressDlg::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);

    if (m_pDIBBack)
    {
        SetWindowPos(NULL, 0, 0, m_pDIBBack->GetWidth(), m_pDIBBack-
>GetHeight(),
                     SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
    }
}

BOOL CProgressDlg::OnEraseBkgnd(CDC* pDC)
{
    CDialog::OnEraseBkgnd(pDC);
}

```

```

    if (m_bFirst)
    {
        m_bFirst = FALSE;
        DownloadFilesList();
    }

    // Make sure we have what we need to do a paint.
    if (!m_pDIBBack)
    {
        TRACE("CProgressDlg: No DIB!\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = pDC->SelectPalette(m_pPal, FALSE);    //
bForceBackground = FALSE
        // dc.RealizePalette();    // we realize in response to
WM_QUERYNEWPALETTE
    }
    m_pDIBBack->Draw(pDC, 0, 0);
    // Select old palette if we altered it.
    if (pPalOld)
        pDC->SelectPalette(pPalOld, FALSE);

    return TRUE;
}

HBRUSH CProgressDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    if (nCtlColor == CTLCOLOR_STATIC)
    {
        pDC->SetBkMode(TRANSPARENT);
        return (HBRUSH)m_NullBrush;
    }
    else
    {
        HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
        return hbr;
    }
}

void CProgressDlg::UpdateControlBackground(const CRect& rcBack)
{
    if (!m_pDIBBack)
        return;
    CRect rc(rcBack);
    InvalidateRect(&rc, TRUE);
}

void CProgressDlg::UpdateDisplay()
{
    CWnd* pWnd = GetDlgItem(IDC_DISPLAY);
    pWnd->ShowWindow(SW_HIDE);
    UpdateControlBackground(RECT_DISPLAY);
}

```

ProgressDlg.cpp

```
        UpdateData(FALSE);          // Write
        pWnd->ShowWindow(SW_SHOW);
    }

void CProgressDlg::UpdateProgress()
{
    CWnd* pWnd = GetDlgItem(IDC_PROGRESS);
    pWnd->ShowWindow(SW_HIDE);
    UpdateControlBackground(RECT_PROGRESS);
    UpdateData(FALSE);          // Write
    pWnd->ShowWindow(SW_SHOW);
}

UINT CProgressDlg::OnNcHitTest(CPoint point)
{
    UINT nHitTest = CDialog::OnNcHitTest(point);
    CPoint pt(point);
    ScreenToClient(&pt);
    if (!RECT_EDIT.PtInRect(pt) &&
        (nHitTest == HTCLIENT) && (::GetAsyncKeyState(MK_LBUTTON) < 0))
        nHitTest = HTCAPTION;
    return nHitTest;
}
```

ProgressDlg.h

```
// ProgressDlg.h : header file
//

#if !defined(AFX_PROGRESSDLG_H__C8A475E9_CD64_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_PROGRESSDLG_H__C8A475E9_CD64_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "UC2Ani/PSButton.h"

////////////////////////////////////
// CProgressDlg dialog
class CDownload;

class CDownInfo
{
public:    // CFileStatus
    CString      m_strFile;
    int          m_nType;    // 0: No Extract, 1: Extract
    LONG         m_lSize;    // The logical size of the file in bytes
    CTime        m_mtime;    // The date and time the file was created.
    BOOL         m_bNew;     // Newly downloaded file
};

class CDIB;

class CProgressDlg : public CDialog
{
// Construction
public:
    CProgressDlg(CWnd* pParent = NULL); // standard constructor
    ~CProgressDlg();

    BOOL RITModified() const { return m_bRITModified; }

// Dialog Data
    //{AFX_DATA(CProgressDlg)
    enum { IDD = IDD_PROGRESS_DIALOG };
    CEdit m_ecMsg;
    CProgressCtrl m_pbProgress;
    CString m_strDisplay;
    CString m_strProgress;
    //}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CProgressDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    //}AFX_VIRTUAL

// Implementation
protected:
    void InitControl(const int nCtrlID, const CRect& rcCtrl);
    void UpdateControlBackground(const CRect& rcBack);

```



```

void        UpdatedDisplay();
void        UpdateProgress();
void        DownloadFilesList();
void        DownloadNewFile();

CDIB*       m_pDIBBack;           // Background frame image
CPalette*   m_pPal;               // main palette
BOOL        m_bPaletteCreated;
CPSButton   m_btnOK;
CPSButton   m_btnCancel;
CBrush      m_NullBrush;
CFont       m_ftMessage;

CDownload*  m_pDownload;
CDownInfo*  m_aDI;
int         m_nDis;
int         m_nCurDI;
CString     m_strHost;
CString     m_strDownloadingFile;
BOOL        m_bFirst;
BOOL        m_bMOD;               // Message of the Day
BOOL        m_bRITModified;      // need to reload stage

// ON_MESSAGE
LRESULT OnBindStart(WPARAM, LPARAM);
LRESULT OnBindFailed(WPARAM, LPARAM);
LRESULT OnBindFileOpened(WPARAM, LPARAM);
LRESULT OnBindFileClosed(WPARAM, LPARAM);
LRESULT OnBindProgressMsg(WPARAM, LPARAM);
LRESULT OnBindProgressBar(WPARAM, LPARAM);
LRESULT OnBindDownloadDone(WPARAM, LPARAM);

// Generated message map functions
//{{AFX_MSG(CProgressDlg)
virtual BOOL OnInitDialog();
virtual void OnOK();
virtual void OnCancel();
afx_msg BOOL OnQueryNewPalette();
afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
afx_msg BOOL OnEraseBkgnd(CDC* pDC);
afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
afx_msg void OnSize(UINT nType, int cx, int cy);
afx_msg UINT OnNcHitTest(CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_PROGRESSDLG_H__C8A475E9_CD64_11D1_80E2_080009B9F339__INCLUDED_)

```

PSFrame.cpp

```
// PSFrame.cpp : implementation file
//

#include "stdafx.h"
#include "resource.h"
#include "PSFrame.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CPSFrame

IMPLEMENT_DYNCREATE(CPSFrame, CMiniFrameWnd)

CPSFrame::CPSFrame()
{
    m_pModelessPropSheet = NULL;
}

CPSFrame::~CPSFrame()
{
}

BEGIN_MESSAGE_MAP(CPSFrame, CMiniFrameWnd)
    //{{AFX_MSG_MAP(CPSFrame)
    ON_WM_CREATE()
    ON_WM_CLOSE()
    ON_WM_SETFOCUS()
    ON_WM_ACTIVATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// CPSFrame message handlers

int CPSFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMiniFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    m_pModelessPropSheet = new CPSMemberInfo(this);
    if (!m_pModelessPropSheet->Create(this, WS_CHILD | WS_VISIBLE, 0))
    {
        delete m_pModelessPropSheet;
        m_pModelessPropSheet = NULL;
        return -1;
    }

    // Resize the mini frame so that it fits around the child property
    // sheet.
    CRect rectClient, rectWindow;
    m_pModelessPropSheet->GetWindowRect(rectClient);

```

```

    rectWindow = rectClient;

    // CMiniFrameWnd::CalcWindowRect adds the extra width and height
    // needed from the mini frame.
    CalcWindowRect(rectWindow);
    SetWindowPos(NULL, rectWindow.left, rectWindow.top,
        rectWindow.Width(), rectWindow.Height(),
        SWP_NOZORDER | SWP_NOACTIVATE);
    m_pModelessPropSheet->SetWindowPos(NULL, 0, 0,
        rectClient.Width(), rectClient.Height(),
        SWP_NOZORDER | SWP_NOACTIVATE);

    return 0;
}

void CPSFrame::OnClose()
{
    // Instead of closing the modeless property sheet, just hide it.
    ShowWindow(SW_HIDE);
}

void CPSFrame::OnSetFocus(CWnd* pOldWnd)
{
    CMiniFrameWnd::OnSetFocus(pOldWnd);
}

void CPSFrame::OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized)
{
    CMiniFrameWnd::OnActivate(nState, pWndOther, bMinimized);

    // Forward any WM_ACTIVATEs to the property sheet...
    // Like the dialog manager itself, it needs them to save/restore the
    focus.
    ASSERT_VALID(m_pModelessPropSheet);

    // Use GetCurrentMessage to get unmodified message data.
    const MSG* pMsg = GetCurrentMessage();
    ASSERT(pMsg->message == WM_ACTIVATE);
    m_pModelessPropSheet->SendMessage(pMsg->message, pMsg->wParam, pMsg->
        lParam);
}

void CPSFrame::PostNcDestroy()
{
    CMiniFrameWnd::PostNcDestroy();
    // delete this;
}

```

PSFrame.h

```
// PSFrame.h : header file
//
// This file contains the mini-frame that controls modeless
// property sheet CPSMemberInfo.

#ifndef __PSFRAME_H__
#define __PSFRAME_H__

#include "PSMemberInfo.h"

////////////////////////////////////
// CPSFrame frame

class CPSFrame : public CMiniFrameWnd
{
    DECLARE_DYNCREATE(CPSFrame)
//Construction
public:
    CPSFrame();

// Attributes
public:
    CPSMemberInfo* m_pModelessPropSheet;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPSFrame)
    protected:
    virtual void PostNcDestroy();
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CPSFrame();

    // Generated message map functions
    //{{AFX_MSG(CPSFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnClose();
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
#endif // __PSFRAME_H__
```

```

// PSJoinChannel.cpp : implementation file
//

#include "stdafx.h"
#include "resource.h"
#include "PSJoinChannel.h"
#include "ResMan.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"
#include "MainFrm.h"
#include "MemberInfo.h"
#include "UC2Doc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

LPCTSTR HTTP = _T("http://");

////////////////////////////////////
// CPSJoinChannel

IMPLEMENT_DYNAMIC(CPSJoinChannel, CPropertySheet)

CPSJoinChannel::CPSJoinChannel(CUC2Doc* pDoc, const int nPageSelect, LPCTSTR
pszCaption, UINT iSelectPage)
    : CPropertySheet(pszCaption, NULL, iSelectPage) /*
IDS_PROPSHT_CAPTION */
{
    TRACE0("CPSJoinChannel::CPSJoinChannel()\n");

    m_pDoc = pDoc;
    m_nPageSelect = nPageSelect;

    m_brNull.CreateStockObject(NULL_BRUSH);
    m_font.CreateFont(-12, 0, 0, 0, FW_BOLD, // NORMAL,
        FALSE, FALSE, 0, // bItalic, bUnderline, cStrikeOut
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH || FF_DONTCARE,
#ifdef _KOREAN
        "±¼, ²Ä¼");
#else
        "Times New Roman");
#endif
    m_pPal = NULL; // Set it NULL before loading DIB successfully
    m_pDIBBack = NULL;

    CString strPath(*gResMan.GetResPath());
/*
    CString strFile(strPath + "U2Login|LoginBk.bmp");
    m_pDIBBack = new CDIB;
    if (!m_pDIBBack->Load(strFile))

```

```

    {
        delete m_pDIBBack;
        m_pDIBBack = NULL;
        return;
    }
    m_pDIBBack->ShiftRGBPercent(0, 256, 100);
*/
CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();

// if (pMF->Is256Palette())
// { // Use mainframe's palette to avoid color flickering
//     m_pPal = pMF->GetPalette();
//     m_pDIBBack->MapColorsToPalette(m_pPal);
//     m_bPaletteCreated = FALSE;
/*
}
else // Use original palette in the file for TRUE color system
{
    // Create the palette from the DIB.
    CDIBPal* pDIBPal;
    pDIBPal = new CDIBPal;
    ASSERT(pDIBPal);
    if (!pDIBPal->Create(m_pDIBBack))
    {
        AfxMessageBox("Failed to create palette from DIB file");
        delete pDIBPal;
    }
    m_pPal = pDIBPal; // type casting to parent class
    m_bPaletteCreated = TRUE;
}

*/
// m_pDIBBack->GetRect(m_rcPSheet);
// m_rcPSheet.InflateRect(14, 49);
/*
// Create the palette from the DIB.
m_pPal = new CDIBPal;
ASSERT(m_pPal);
if (!m_pPal->Create(m_pDIBBack))
{
    AfxMessageBox("Failed to create palette from DIB file");
    delete m_pPal;
    m_pPal = NULL;
}

*/
// Add all of the property pages here. Note that
// the order that they appear in here will be
// the order they appear in on screen. By default,
// the first page of the set is the active one.
// One way to make a different property page the
// active one is to call SetActivePage().

TRACE0("\tAddPage(&m_ActorListPage);\n");

if (nPageSelect & PS_ACTOR_PAGE)
    AddPage(&m_ActorListPage);
#ifdef _MALL
if (nPageSelect & PS_JOIN_PAGE)
    AddPage(&m_ShoppingCartPage);
if (nPageSelect & PS_CREATE_PAGE)

```

```

        AddPage (&m_PaymentInfoPage);
#else
    if (nPageSelect & PS_JOIN_PAGE)
        AddPage (&m_JoinChannelPage);
    if (nPageSelect & PS_CREATE_PAGE)
        AddPage (&m_CreateChannelPage);
#endif
}

CPSJoinChannel::~CPSJoinChannel()
{
    TRACE0 ("CPSJoinChannel::~CPSJoinChannel()\n");
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal && m_bPaletteCreated)
        delete m_pPal;
}

BEGIN_MESSAGE_MAP(CPSJoinChannel, CPropertySheet)
//{{AFX_MSG_MAP(CPSJoinChannel)
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPSJoinChannel message handlers
// To be shared in child property pages
BOOL CPSJoinChannel::OnPageEraseBknd(CDC* pDC)
{
    // Make sure we have what we need to do a paint.
    if (!GetDIBBack())
    {
        //
        TRACE("CPSJoinChannel: No DIB!\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    CPalette* pPal = GetPalette();
    if (pPal)
    {
        pPalOld = pDC->SelectPalette(pPal, FALSE); //
        bForceBackground = FALSE
        // dc.RealizePalette(); // we realize in response to
        WM_QUERYNEWPALETTE
    }
    if (m_pDIBBack)
    {
        m_pDIBBack->Draw(pDC, 0, 0, m_pDIBBack->GetWidth(), m_pDIBBack->GetHeight(),
            12, 12);
    }
    // Select old palette if we altered it.
    if (pPalOld)
        pDC->SelectPalette(pPalOld, FALSE);
}

```

```

        return TRUE;
    }

    BOOL CPSJoinChannel::OnQueryNewPalette()
    {
        TRACE("CPSJoinChannel::OnQueryNewPalette\n");
        // if (m_pDoumi && m_pDoumi->GetDIBPal())
        if (m_pPal)
        {
            CDC* pdc = GetDC();
            CPalette* pPalOld = pdc->SelectPalette((CPalette*)m_pPal, FALSE);
            // foreground
            UINT u = pdc->RealizePalette();
            if (pPalOld)
                pdc->SelectPalette(pPalOld, FALSE);
            ReleaseDC(pdc);
            if (u)
            {
                // Some colors changed so we need to do a repaint.
                Invalidate(); // Repaint the lot.
                return TRUE; // Say we did something.
            }
        }
        return FALSE; // Say we did nothing.
    }

    void CPSJoinChannel::OnPaletteChanged(CWnd* pFocusWnd)
    {
        TRACE("CPSJoinChannel::OnPaletteChanged\n");
        if (pFocusWnd != this)
            OnQueryNewPalette();
    }

    /*
    void CPSJoinChannel::OnSize(UINT nType, int cx, int cy)
    {
        TRACE("CPSJoinChannel::OnSize(%d,%d,%d)\n", nType, cx, cy);
        CPropertySheet::OnSize(nType, cx, cy);

        if (m_pDIBBack)
        {
            SetWindowPos(NULL, 0, 0, m_rcPSheet.Width(), m_rcPSheet.Height(),
                          SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
        }
    }
    */

    BOOL CPSJoinChannel::OnInitDialog()
    {
        TRACE("CPSJoinChannel::OnInitDialog() => 0x%x\n", this);

        /*
        SetIcon(m_hIcon, TRUE);
        SetIcon(m_hIcon, FALSE);
        CRect rc;
        CWnd* pW = (CWnd*)GetDlgItem(IDOK);
        if (pW)
        {

```



```

        pW->GetClientRect(&rc);
        pW->SetWindowPos(NULL, rcPSheet.left, rcCtrl.top,
                        rcCtrl.Width(), rcCtrl.Height(),
                        SWP_NOZORDER | SWP_NOACTIVATE);
#ifdef _ENGLISH
        pW->SetWindowText("OK!");
#endif
    }
    pW = (CWnd*)GetDlgItem(IDCANCEL);
    if (pW)
        pW->SetWindowText("Cancel");
    pW = (CWnd*)GetDlgItem(IDHELP);
    if (pW)
        pW->SetWindowText("Help");
    pW = (CWnd*)GetDlgItem(0x3021);    // Apply
    if (pW)
        pW->SetWindowText("Apply");
*/
#ifdef _MALL
    ((CWnd*)GetDlgItem(IDOK))->SetWindowText("OK");
    ((CWnd*)GetDlgItem(IDCANCEL))->SetWindowText("Cancel");
#endif
    CWnd* pW = (CWnd*)GetDlgItem(IDHELP);
    if (pW)
        pW->ShowWindow(SW_HIDE);
    pW = (CWnd*)GetDlgItem(0x3021);    // Apply
    if (pW)
        pW->ShowWindow(SW_HIDE);
//    pW->EnableWindow(FALSE);
    return CPropertySheet::OnInitDialog();
}

void CPSJoinChannel::InitControl(const int nCtrlID, const CRect& rcCtrl)
{
    CWnd* pW = GetDlgItem(nCtrlID);
    ASSERT(pW);
    pW->SetWindowPos(NULL, rcCtrl.left, rcCtrl.top,
                    rcCtrl.Width(), rcCtrl.Height(),
                    SWP_NOZORDER | SWP_NOACTIVATE);
    pW->SetFont(&m_font, FALSE);    // do not Redraw
}

BOOL CPSJoinChannel::GetChannelInfo(EC_CHANNELINFO& ecInfo)
{
    ::ZeroMemory(&ecInfo, sizeof(EC_CHANNELINFO));

    CString* pS;
    int len;
    // N.B. You should not pass a pointer to objects in this stack!
    if (IsCreateChannel()) {
        pS = m_CreateChannelPage.GetStageName();
        len = pS->GetLength();
        if (!len)
            return FALSE;
        ecInfo.szName = pS->GetBuffer(len+1); //
[ep0/0000cntrl]{0,0,0}test
        ecInfo.dwType = m_CreateChannelPage.IsPublic()

```

```

CS_CHANNEL_ALLOWANON */
                                ? CS_CHANNEL_PUBLIC /* |
                                : CS_CHANNEL_PROTECTED; /* |
CS_CHANNEL_ALLOWANON */
    }
    else { // Join the Channel
        // m_strStageID will be the selected channel
        pS = &m_JoinChannelPage.m_strStageID;

        m_CreateChannelPage.m_strTopic = *pS;
        len = pS->GetLength();
        if (!len) { // No room is selected: Consider as a MUD mode // syc
0706
            // m_JoinChannelPage.GetLastStageID(); // syc 0715
            // gResMan.GetStageTitle(m_CreateChannelPage.m_strTopic);
            // = UC2MUD_TOPIC; // syc 0715
            return FALSE; // syc 0715
        }
        ecInfo.szName = pS->GetBuffer(len+1);

        if (m_JoinChannelPage.IsMUD()) {
            gResMan.GetStageTitle(m_CreateChannelPage.m_strTopic);
            // = UC2MUD_TOPIC;
        }

        if (m_JoinChannelPage.m_strPassword.IsEmpty()) {
            ecInfo.dwType = CS_CHANNEL_PUBLIC; /* |
CS_CHANNEL_ALLOWANON */
            m_CreateChannelPage.m_strPassword.Empty();
        }
        else {
            ecInfo.dwType = CS_CHANNEL_PROTECTED; /* |
CS_CHANNEL_ALLOWANON */
            m_CreateChannelPage.m_strPassword =
m_JoinChannelPage.m_strPassword;
        }

        pS = &m_CreateChannelPage.m_strTopic;
        // If the topic and Stage ID is same then Find out the topic
        //from Stage ID
        if (m_CreateChannelPage.m_strTopic == m_JoinChannelPage.m_strStageID)
        {
            CString str(*pS);
            int Pos = str.ReverseFind(' ');
            m_CreateChannelPage.m_strTopic = str.Mid(Pos+1, str.GetLength() -
1);
        }

        pS = &m_CreateChannelPage.m_strTopic;

        len = pS->GetLength();
        ecInfo.szTopic = pS->GetBuffer(len+1);

        if (m_CreateChannelPage.m_strPassword.IsEmpty()) {
            ecInfo.szPass = _T("");
        }
        else {

```

```

        pS = &m_CreateChannelPage.m_strPassword;
        len = pS->GetLength();
        ecInfo.szPass      = pS->GetBuffer(len+1);
    }

    ecInfo.cUsersMax = MAX_MEMBERS_IN_CHANNEL;    // limit # of members
in a channel
    ecInfo.dwFlags    = CS_CHANNEL_FLAG_MICONLY;

//    CPropertySheet::OnClose();
    return TRUE;

}

void CPSJoinChannel::SetMemberInfo(CMemberInfo& mi)
{
    m_ActorListPage.m_nCharID      = mi.GetCharID();
    m_ActorListPage.m_nAge         = mi.GetAge();
    m_ActorListPage.m_nSex         = mi.GetSex();
//    m_ActorListPage.m_strProfile  = *mi.GetProfile();
//    m_ActorListPage.m_sName       = *mi.GetRealName();
//    m_ActorListPage.m_strChatID   = *mi.GetNick();
//    m_ActorListPage.m_sHomePage   = *mi.GetHyperlink();
//    if (m_ActorListPage.m_sHomePage.IsEmpty())
//        m_ActorListPage.m_sHomePage = HTTP;
    int nVer = mi.GetVersion();
//    m_ActorListPage.m_sVersion.Format("V%d.%02d ", nVer / 100, nVer % 100);
}

void CPSJoinChannel::GetMemberInfo(CMemberInfo& mi) const
{
    mi.SetCharID(m_ActorListPage.m_nCharID);
    mi.SetAge(m_ActorListPage.m_nAge);
    mi.SetSex(m_ActorListPage.m_nSex);
    mi.SetBubbleKind(m_ActorListPage.m_nSex); // Later I'll change this...
//    mi.SetProfile(m_ActorListPage.m_strProfile);
//    mi.SetRealName(m_ActorListPage.m_strName);
//    mi.SetNick(m_ActorListPage.m_strChatID);
//    if (!m_ActorListPage.m_sHomePage.IsEmpty() &&
//        (lstrcmpi(m_ActorListPage.m_strHomePage, HTTP) != 0))
//        mi.SetHyperlink(m_ActorListPage.m_strHomePage);
}

BOOL CPSJoinChannel::ActivateCreateChannelPage()
{
    if (!(m_nPageSelect & PS_CREATE_PAGE))
        AddPage(&m_CreateChannelPage);
    return SetActivePage(&m_CreateChannelPage);
}

BOOL CPSJoinChannel::ActivateJoinChannelPage()
{
    if (!(m_nPageSelect & PS_JOIN_PAGE))
        AddPage(&m_JoinChannelPage);
    return SetActivePage(&m_JoinChannelPage);
}

```

PSJoinChannel.h

```
// PSJoinChannel.h : header file
//
// This class defines custom modal property sheet
// CPSJoinChannel.

#ifndef __PSJOINCHANNEL_H__
#define __PSJOINCHANNEL_H__

#ifdef _MALL
#include "PPShoppingCart.h"
#include "PPPaymentInfo.h"
#endif
#include "PPCreateChannel.h"
#include "PPChannel.h"
#include "PPActor.h"

#include "PPMyInfo.h"
#include "UC2CS.h"          // EC_CHANNELINFO
#include "UC2Doc.h"         // GetSocket()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CPSJoinChannel

class CDIB;
class CMemberInfo;
class CUC2Socket;

enum PAGE_SELECT
{
    PS_ACTOR_PAGE      = 0x0001,
    PS_JOIN_PAGE       = 0x0002,
    PS_CREATE_PAGE     = 0x0004,
    PS_ALL_PAGES       = PS_ACTOR_PAGE | PS_JOIN_PAGE | PS_CREATE_PAGE
};

class CPSJoinChannel : public CPropertySheet
{
    DECLARE_DYNAMIC(CPSJoinChannel)

// Construction
public:
    CPSJoinChannel(CUC2Doc* pDoc, const int nPageSelect, LPCTSTR
pszcaption, UINT iSelectPage=0);

// Attributes
public:
    CDIB*          GetDIBBack()          { return m_pDIBBack; }
    CPalette*      GetPalette()          { return m_pPal; }
    CFont*         GetFont()              { return &m_font; }
    CBrush*        GetNullBrush()        { return &m_brNull; }

    BOOL          IsListOnly() const      { return (m_nPageSelect ==
PS_JOIN_PAGE); }
    BOOL          IsCreateChannel() const
                { return ((m_nPageSelect & PS_CREATE_PAGE) &&
!m_CreateChannelPage.m_strTopic.IsEmpty()); }
};
```

PSJoinChannel.h

```

        CUC2Doc*          GetDocument()      { return m_pDoc; }
        CUC2Socket*       GetSocket()       { return m_pDoc->GetSocket(); }

        CPPActor          m_ActorListPage;
//      CPPMyInfo         m_ActorListPage;

#ifdef _MALL
        CPPShoppingCart   m_ShoppingCartPage;
        CPPPaymentInfo    m_PaymentInfoPage;
#endif
        CPPChannel        m_JoinChannelPage;
        CPPCreateChannel  m_CreateChannelPage;
        EC_CHANNELINFO    m_ecInfo;

// Operations
public:
        BOOL              OnPageEraseBkgnd(CDC* pDC);
        void              InitControl(const int nCtrlID, const CRect& rcCtrl);
        BOOL              ActivateCreateChannelPage();
        BOOL              ActivateJoinChannelPage();
        BOOL              GetChannelInfo(EC_CHANNELINFO& ecInfo);
        void              SetMemberInfo(CMemberInfo& mi);
        void              GetMemberInfo(CMemberInfo& mi) const;

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CPSJoinChannel)
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CPSJoinChannel();

// Generated message map functions
protected:
        CDIB*             m_pDIBBack;          // Background frame image
        CPalette*          m_pPal;             // main palette
        BOOL               m_bPaletteCreated;
//      CRect             m_rcPSheet;
        CBrush             m_brNull;
        CFont              m_font;
        int                m_nPageSelect;      // enum PAGE_SELECT
        CUC2Doc*           m_pDoc;            // To get pointers to Socket
        //{{AFX_MSG(CPSJoinChannel)
        afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
        afx_msg BOOL OnQueryNewPalette();
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

#endif      // __PSJOINCHANNEL_H__

```

PSMemberInfo.cpp

```
// PSMemberInfo.cpp : implementation file
//

#include "stdafx.h"
#include "resource.h"
#include "PSMemberInfo.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CPSMemberInfo

IMPLEMENT_DYNAMIC(CPSMemberInfo, CPropertySheet)

CPSMemberInfo::CPSMemberInfo(CWnd* pParent)
    : CPropertySheet(IDS_PROPSHT_CAPTION, pParent)
{
    // Add all of the property pages here. Note that
    // the order that they appear in here will be
    // the order they appear in on screen. By default,
    // the first page of the set is the active one.
    // One way to make a different property page the
    // active one is to call SetActivePage().

    AddPage(&m_Page1);
    AddPage(&m_Page2);
}

CPSMemberInfo::~CPSMemberInfo()
{
}

BEGIN_MESSAGE_MAP(CPSMemberInfo, CPropertySheet)
    //{{AFX_MSG_MAP(CPSMemberInfo)
    // NOTE - the ClassWizard will add and remove mapping macros
    here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPSMemberInfo message handlers

void CPSMemberInfo::PostNcDestroy()
{
    CPropertySheet::PostNcDestroy();
    delete this;
}
```

PSMemberInfo.h

```
// PSMemberInfo.h : header file
//
// CPSMemberInfo is a modeless property sheet that is
// created once and not destroyed until the application
// closes. It is initialized and controlled from
// CPSFrame.

#ifndef __PSMEMBERINFO_H__
#define __PSMEMBERINFO_H__

#include "PPMemberInfo1.h"

////////////////////////////////////

// CPSMemberInfo

class CPSMemberInfo : public CPropertySheet
{
    DECLARE_DYNAMIC(CPSMemberInfo)

// Construction
public:
    CPSMemberInfo(CWnd* pWndParent = NULL);

// Attributes
public:
    CPPMemberInfo1 m_Page1;
    CPPMemberInfo2 m_Page2;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPSMemberInfo)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CPSMemberInfo();
    virtual void PostNcDestroy();

// Generated message map functions
protected:
    //{{AFX_MSG(CPSMemberInfo)
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

#endif // __PSMEMBERINFO_H__
```

```

// PSOtherInfo.cpp : implementation file
//

#include "stdafx.h"
#include "resource.h"
#include "PSOtherInfo.h"
#include "ResMan.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"
#include "MainFrm.h"
#include "MemberInfo.h"
#include "UC2Doc.h"
#include "Parser.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;
extern CParser gParser;

////////////////////////////////////
// CPSOtherInfo

IMPLEMENT_DYNAMIC(CPSOtherInfo, CPropertySheet)

CPSOtherInfo::CPSOtherInfo(CUC2Doc* pDoc, const int nPageSelect, LPCTSTR
pszCaption, UINT iSelectPage)
    : CPropertySheet(pszCaption, NULL, iSelectPage) /*
IDS_PROPSHT_CAPTION */
{
    TRACE0("CPSOtherInfo::CPSOtherInfo()\n");

    m_pDoc = pDoc;
    m_nPageSelect = nPageSelect;

    m_brNull.CreateStockObject(NULL_BRUSH);
    m_font.CreateFont(-12, 0, 0, 0, FW_BOLD, // NORMAL,
        FALSE, FALSE, 0, // bItalic, bUnderline, cStrikeOut
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH || FF_DONTCARE,
#ifdef _KOREAN
        "±¼, ²Ã¼");
#else
        "Times New Roman");
#endif
    m_pPal = NULL; // Set it NULL before loading DIB successfully
    m_pDIBBack = NULL;

    CString strPath(gResMan.GetResPath());

    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    // Add all of the property pages here. Note that
    // the order that they appear in here will be

```



```

// the order they appear in on screen. By default,
// the first page of the set is the active one.
// One way to make a different property page the
// active one is to call SetActivePage().

TRACE0("\tAddPage(&m_ActorListPage);\n");

if (nPageSelect & PS_OTHERINFO_PAGE)
    AddPage(&m_OtherInfoPage);

// if (nPageSelect & PS_OTHERADDR_PAGE)
//     AddPage(&m_UserAddrPage);

// if (nPageSelect & PS_OTHERPROF_PAGE)
//     AddPage(&m_UserProfilePage);

// if (nPageSelect & PS_OTHERPIC_PAGE)
//     AddPage(&m_UserPicPage);
}

CPSOtherInfo::~CPSOtherInfo()
{
    TRACE0("CPSOtherInfo::~CPSOtherInfo()\n");
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal && m_bPaletteCreated)
        delete m_pPal;
}

BEGIN_MESSAGE_MAP(CPSOtherInfo, CPropertySheet)
    //{AFX_MSG_MAP(CPSOtherInfo)
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPSOtherInfo message handlers
// To be shared in child property pages
BOOL CPSOtherInfo::OnPageEraseBkgnd(CDC* pDC)
{
    // Make sure we have what we need to do a paint.
    if (!GetDIBBack())
    {
        //
        TRACE("CPSOtherInfo: No DIB!\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    CPalette* pPal = GetPalette();
    if (pPal)
    {
        pPalOld = pDC->SelectPalette(pPal, FALSE); //
        bForceBackground = FALSE

```

```

        // dc.RealizePalette();    // we realize in response to
WM_QUERYNEWPALETTE
    }
    if (m_pDIBBack)
    {
        m_pDIBBack->Draw(pDC, 0, 0, m_pDIBBack->GetWidth(), m_pDIBBack-
>GetHeight(),
                                12, 12);
    }
    // Select old palette if we altered it.
    if (pPalOld)
        pDC->SelectPalette(pPalOld, FALSE);

    return TRUE;
}

BOOL CPSOtherInfo::OnQueryNewPalette()
{
    TRACE("CPSOtherInfo::OnQueryNewPalette\n");
    // if (m_pDoumi && m_pDoumi->GetDIBPal())
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette((CPalette*)m_pPal, FALSE);
        // foreground
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
        if (u)
        {
            // Some colors changed so we need to do a repaint.
            Invalidate();    // Repaint the lot.
            return TRUE;    // Say we did something.
        }
    }
    return FALSE;    // Say we did nothing.
}

void CPSOtherInfo::OnPaletteChanged(CWnd* pFocusWnd)
{
    TRACE("CPSOtherInfo::OnPaletteChanged\n");
    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CPSOtherInfo::OnInitDialog()
{
    TRACE("CPSOtherInfo::OnInitDialog() => 0x%x\n", this);

    CWnd* pW = (CWnd*)GetDlgItem(IDHELP);
    if (pW)
        pW->ShowWindow(SW_HIDE);
    pW = (CWnd*)GetDlgItem(0x3021);    // Apply
    if (pW)
        pW->ShowWindow(SW_HIDE);
    //
    pW->EnableWindow(FALSE);
}

```

```

        return CPropertySheet::OnInitDialog();
    }

void CPSOtherInfo::InitControl(const int nCtrlID, const CRect& rcCtrl)
{
    CWnd* pW = GetDlgItem(nCtrlID);
    ASSERT(pW);
    pW->SetWindowPos(NULL, rcCtrl.left, rcCtrl.top,
                     rcCtrl.Width(), rcCtrl.Height(),
                     SWP_NOZORDER | SWP_NOACTIVATE);
    pW->SetFont(&m_font, FALSE); // do not Redraw
}

/*void CPSOtherInfo::SetMemberProfile(CMemberInfo& mi)
{
    m_UserProfilePage.m_sProfile = *mi.GetProfile();
}

void CPSOtherInfo::SetMemberAddress(CMemberInfo& mi)
{
    m_UserAddrPage.m_sAddrStreet = *mi.GetAddrStreet();
    m_UserAddrPage.m_sAddrCity = *mi.GetAddrCity();
    m_UserAddrPage.m_sAddrZip = *mi.GetAddrZip();
    m_UserAddrPage.m_sAddrState = *mi.GetAddrState();
    m_UserAddrPage.m_nCountry = mi.GetCountry();

    m_UserAddrPage.m_sTelHome = *mi.GetTelHome();
    m_UserAddrPage.m_sTelWork = *mi.GetTelWork();
    m_UserAddrPage.m_sTelMobile = *mi.GetTelMobile();
    m_UserAddrPage.m_sTelFax = *mi.GetTelFax();
    m_UserAddrPage.m_sTelOther = *mi.GetTelOther();
}
*/

void CPSOtherInfo::SetMemberInfo(CString* str)
{
    TRACE("\nCPSOtherInfo::SetMemberInfo() on string: %s", str);

    int nAge;
    int nSex;
    int nEthnic;
    int nWork;
    int nLanguage;

    gParser.GetValueRightToken(m_OtherInfoPage.m_sFName, ARG_TOKEN);
    gParser.GetValueRightToken(m_OtherInfoPage.m_sLName, ARG_TOKEN);
    gParser.GetValueRightToken(nAge, ARG_TOKEN);
    gParser.GetValueRightToken(nSex, ARG_TOKEN);
    gParser.GetValueRightToken(nEthnic, ARG_TOKEN);
    gParser.GetValueRightToken(nLanguage, ARG_TOKEN);
    gParser.GetValueRightToken(nWork, ARG_TOKEN);

    m_OtherInfoPage.m_sAge.Format("%d", nAge);

```

```

        m_OtherInfoPage.m_sSex.Format("%s", SexTable[nSex]);
        m_OtherInfoPage.m_sEthnic.Format("%s", EthnicTable[nEthnic]);
        m_OtherInfoPage.m_sLanguage.Format("%s", LanguageTable[nLanguage]);
        m_OtherInfoPage.m_sWork.Format("%s", WorkTable[nWork]);

        gParser.GetValueRightToken(m_OtherInfoPage.m_sEmail, ARG_TOKEN);
    }

/*
void CPSOtherInfo::GetMemberProfile(CMemberInfo& mi) const
{
    mi.SetProfile(    m_UserProfilePage.m_sProfile);
}

void CPSOtherInfo::GetMemberAddress(CMemberInfo& mi) const
{
    mi.SetAddrStreet( m_UserAddrPage.m_sAddrStreet);
    mi.SetAddrCity(   m_UserAddrPage.m_sAddrCity);
    mi.SetAddrState(  m_UserAddrPage.m_sAddrState);
    mi.SetAddrZip(    m_UserAddrPage.m_sAddrZip);

    mi.SetCountry(    m_UserAddrPage.m_nCountry);

    mi.SetTelHome(    m_UserAddrPage.m_sTelHome);
    mi.SetTelWork(    m_UserAddrPage.m_sTelWork);
    mi.SetTelMobile(  m_UserAddrPage.m_sTelMobile);
    mi.SetTelFax(     m_UserAddrPage.m_sTelFax);
    mi.SetTelOther(   m_UserAddrPage.m_sTelOther);
}

void CPSOtherInfo::GetMemberInfo(CMemberInfo& mi) const
{
    mi.SetCharID(     m_MyInfoPage.m_nCharID);
    mi.SetAge(        m_MyInfoPage.m_nAge);
    mi.SetSex(        m_MyInfoPage.m_nSex);
    mi.SetBubbleKind(m_MyInfoPage.m_nSex);    // Later I'll change this...
    // mi.SetProfile(  m_MyInfoPage.m_strProfile);

    CString rn = m_MyInfoPage.m_sFName;
    rn         += " ";
    rn         += m_MyInfoPage.m_sLName;
    mi.SetRealName(   rn );
    mi.SetNick(       m_MyInfoPage.m_sAlias);
    if (!m_MyInfoPage.m_sHomePage.IsEmpty() &&
        (lstrcmpi(m_MyInfoPage.m_sHomePage, HTTP) != 0))
        mi.SetHyperlink(m_MyInfoPage.m_sHomePage);

    mi.SetEmail(      m_MyInfoPage.m_sEmail);
    mi.SetEthnic(      m_MyInfoPage.m_nEthnic);
    mi.SetLanguage(    m_MyInfoPage.m_nLanguage);
    mi.SetFName(       m_MyInfoPage.m_sFName);
    mi.SetLName(       m_MyInfoPage.m_sLName);
    mi.SetWork(        m_MyInfoPage.m_nWork);

```

```
}

BOOL CPSOtherInfo::ActivateCreateChannelPage()
{
    if (!(m_nPageSelect & PS_CREATE_PAGE))
        AddPage(&m_CreateChannelPage);
    return SetActivePage(&m_CreateChannelPage);
}

BOOL CPSOtherInfo::ActivateJoinChannelPage()
{
    if (!(m_nPageSelect & PS_JOIN_PAGE))
        AddPage(&m_JoinChannelPage);
    return SetActivePage(&m_JoinChannelPage);
}
*/
```

PSOtherInfo.h

```
// PSOtherInfo.h : header file
//
// This class defines custom modal property sheet
// CPSOtherInfo.

#ifndef __PSOtherInfo_H__
#define __PSOtherInfo_H__

#ifdef _MALL
#include "PPShoppingCart.h"
#include "PPPaymentInfo.h"
#endif
#include "PPCreateChannel.h"
#include "PPChannel.h"
#include "PPActor.h"
#include "PPMyInfo.h"

#include "PPMyAddress.h"
// #include "PPUserPic.h"
// #include "PPUserProfile.h"

#include "PPOtherInfo.h"

#include "UC2CS.h"          // EC_CHANNELINFO
#include "UC2Doc.h"         // GetSocket()

////////////////////////////////////
// CPSOtherInfo

class CDIB;
class CMemberInfo;
class CUC2Socket;

enum PAGE_SELECT
{
    PS_OTHERINFO_PAGE          = 0x0001,
    PS_OTHERADDR_PAGE          = 0x0002,
    PS_OTHERPROF_PAGE          = 0x0004,
    // PS_USERPIC_PAGE          = 0x0004,

    PS_ALL_PAGES                = PS_OTHERINFO_PAGE | PS_OTHERADDR_PAGE |
    PS_OTHERPROF_PAGE
};

class CPSOtherInfo : public CPropertySheet
{
    DECLARE_DYNAMIC(CPSOtherInfo)

// Construction
public:
    CPSOtherInfo(CUC2Doc* pDoc, const int nPageSelect, LPCTSTR pszCaption,
        UINT iSelectPage=0);

// Attributes
public:
    CDIB*          GetDIBBack()          { return m_pDIBBack; }

```

PSOtherInfo.h

```

        CPalette*      GetPalette()          { return m_pPal; }
        CFont*         GetFont()             { return &m_font; }
        CBrush*        GetNullBrush()        { return &m_brNull; }

//      BOOL          IsListOnly() const     { return (m_nPageSelect ==
PS_JOIN_PAGE); }
        CUC2Doc*       GetDocument()         { return m_pDoc; }
//      CUC2Socket*   GetSocket()           { return m_pDoc->GetSocket(); }

//      CPPActor      m_ActorListPage;

        CPPOtherInfo   m_OtherInfoPage;
//      CPPOtherPic   m_OtherPicPage;
//      CPPOtherProfile m_OtherProfilePage;
//      CPPOtherAddr   m_OtherAddrPage;

// Operations
public:
        BOOL          OnPageEraseBkgnd(CDC* pDC);
        void          InitControl(const int nCtrlID, const CRect& rcCtrl);
//      BOOL          ActivateCreateChannelPage();
//      BOOL          ActivateJoinChannelPage();
//      BOOL          GetChannelInfo(EC_CHANNELINFO& ecInfo);
        void          SetMemberInfo(CString* str);
        void          GetMemberInfo(CMemberInfo& mi) const;

        void          SetMemberProfile(CMemberInfo& mi);
        void          SetMemberAddress(CMemberInfo& mi);

        void          GetMemberProfile(CMemberInfo& mi) const;
        void          GetMemberAddress(CMemberInfo& mi) const;

// Overrides
        // ClassWizard generated virtual function overrides
        //{AFX_VIRTUAL(CPSOtherInfo)
        //{AFX_VIRTUAL

// Implementation
public:
        virtual ~CPSOtherInfo();

// Generated message map functions
protected:
        CDIB*         m_pDIBBack;          // Background frame image
        CPalette*      m_pPal;              // main palette
        BOOL          m_bPaletteCreated;
//      CRect         m_rcPSheet;
        CBrush         m_brNull;
        CFont          m_font;
        int            m_nPageSelect;      // enum PAGE_SELECT
        CUC2Doc*       m_pDoc;              // To get pointers to Socket
        //{AFX_MSG(CPSOtherInfo)
        afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
        afx_msg BOOL OnQueryNewPalette();
        virtual BOOL OnInitDialog();

```

PSOtherInfo.h

```
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

static char* SexTable [] = {
    "?",
    "Male",
    "Female"
};

static char* EthnicTable [] = {
    "",
    "African",
    "Asian",
    "Bi/Multi-ethnic",
    "Caucasian",
    "Latino/Hispanic",
    "Middle Eastern",
    "South East Asian",
    "Other"
};

static char* WorkTable [] = {
    "",
    "Academic",
    "Actor/Actress",
    "Administrative",
    "Art/Entertainment",
    "College Student",
    "Community & Social",
    "Computers",
    "Education",
    "Engineering",
    "Financial Services",
    "Government",
    "High School Student",
    "Home",
    "Law",
    "Managerial",
    "Manufacturing",
    "Medical/Health",
    "Military",
    "Musician",
    "Non-Government Organization",
    "Other Services",
    "Professional",
    "Retail",
    "Retired",
    "Science & Research",
    "Sports",
    "Technical",
    "University Student",
    "Web Technology"
};
```



```
static char* LanguageTable [] = {
    "",
    "English",
    "Cantonese",
    "German",
    "Hakka",
    "Hokkien",
    "Italian",
    "Korean",
    "Japanese",
    "Mandarin",
    "Russian",

    "Afrikaans",
    "Arabic",
    "Assamese",
    "Awadhi",
    "Bahasa Indonesia",
    "Bambara",
    "Basque",
    "Bengali",
    "Bhojpur",
    "Breton",
    "Burmese",
    "Creole",
    "Croatian",
    "Czech",
    "Dutch",
    "Esperanto",
    "Estonian",
    "Farsi",
    "Finnish",
    "Gaelic",
    "Greek",
    "Gujarati",
    "Hawaiian",
    "Hebrew",
    "Hindi",
    "Hungarian",
    "Maithili",
    "Maltese",
    "Maori",
    "Native American",
    "Norwegian",
    "Nyanja",
    "Panjabi",
    "Polish",
    "Portuguese",
    "Quechua",
    "Romanian",
    "Rund",
    "Shanghainese",
    "Shona",
    "Swahi",
    "Swat",
    "Swedish",
```

PSOtherInfo.h

```
"Tagalog",  
"Telugu",  
"Thai",  
"Tsonga",  
"Urdu",  
"Welsh",  
"Xhosa",  
"Yiddish",  
"Zulu"  
};
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```
#endif      // __PSOtherInfo_H__
```

```
=====
MICROSOFT FOUNDATION CLASS LIBRARY : UC2
=====
```

AppWizard has created this UC2 application for you. This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that make up your UC2 application.

UC2.h

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CUC2App application class.

UC2.cpp

This is the main application source file that contains the application class CUC2App.

UC2.rc

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

res\UC2.ico

This is an icon file, which is used as the application's icon. This icon is included by the main resource file UC2.rc.

res\UC2.rc2

This file contains resources that are not edited by Microsoft Developer Studio. You should place all resources not editable by the resource editor in this file.

UC2.clw

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

```
////////////////////////////////////
```

For the main frame window:

MainFrm.h, MainFrm.cpp

These files contain the frame class CMainFrame, which is derived from CFrameWnd and controls all SDI frame features.

res\Toolbar.bmp

This bitmap file is used to create tiled images for the toolbar. The initial toolbar and status bar are constructed in the CMainFrame class. Edit this toolbar bitmap along with the array in MainFrm.cpp to add more toolbar buttons.

ReadMe.txt

////////////////////////////////////////////////////////////////

AppWizard creates one document type and one view:

UC2Doc.h, UC2Doc.cpp - the document

These files contain your CUC2Doc class. Edit these files to add your special document data and to implement file saving and loading (via CUC2Doc::Serialize).

UC2View.h, UC2View.cpp - the view of the document

These files contain your CUC2View class.
CUC2View objects are used to view CUC2Doc objects.

////////////////////////////////////////////////////////////////

Help Support:

MakeHelp.bat

Use this batch file to create your application's Help file, hlp\UC2.hlp.

hlp\UC2.hpj

This file is the Help Project file used by the Help compiler to create your application's Help file.

hlp*.bmp

These are bitmap files required by the standard Help file topics for Microsoft Foundation Class Library standard commands.

hlp*.rtf

This file contains the standard help topics for standard MFC commands and screen objects.

////////////////////////////////////////////////////////////////

Other standard files:

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header (PCH) file named UC2.pch and a precompiled types file named StdAfx.obj.

Resource.h

This is the standard header file, which defines new resource IDs. Microsoft Developer Studio reads and updates this file.

////////////////////////////////////////////////////////////////

Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you should add to or customize.

If your application uses MFC in a shared DLL, and your application is in a language other than the operating system's current language, you will need to copy the corresponding localized resources MFC40XXX.DLL from the Microsoft Visual C++ CD-ROM onto the system or system32 directory, and rename it to be MFCLOC.DLL. ("XXX" stands for the language abbreviation. For example, MFC40DEU.DLL contains resources translated to German.) If you don't do this, some of the UI elements of your application will remain in the

language of the operating system.

////////////////////////////////////

```

//      ResMan: Resource Manager
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT.COM
//=====
//
/* =====
; "resrc000.rit" for resrc000.bmc
;=====
; Resource Info Specification
; RIT: Resource Information Table
; Specification written by Soomin Kim
; Feb 1998, Samsung SDS
;=====
; filename=[(col,row)][,(cz.x,cz.y)][,nickname];
; filename = resource type flag(1) + name(4) + serial(3);
; resource type flag = t(TILE), s(STATIC), i(ANI), a(AVATAR), w(WAVE),
m(MIDI)
; (col,row) = # of cells in (column, row); for STATIC this attribute is
omitted
; (cz.x,cz.y) = center of z-order in (x, y) pixel position from the left-
top
; nickname = nickname for actor

// Think of tile as a PhasedSprite
#TIL= // TILE
{
    tcity001=(1,4);
    ...
}
#STT= // STATIC Sprites
{
    ccity002=(23,28);          // Earth Position
    ...
}
#ANI= // ANIMATED Sprites
{
    wroom001=(1,17),(34,8);      // 34,22
    iadv000=(1,3),(123,321);    // not exist
    ...
}
#AVT= // AVATAR
{
    aman_001=(11,4),(27,45),°Å°İ;
    ...
}
#WAV=
{
    wwhip;          // wwhip.wav
    ...
}
#MID=
{
    mtest;          // mtest.mid
    ...
}

```

```

}

// Comments
// Resources are sequentially numbered according to the order in RIT.
// BMP file merger will read this info and then load each bitmap file
// with the same name as it's id (+.bm).

; ACTOR SECTION
; (*i): - change orientation, * 50%, / 25%, | 12%, ~ 0%, ^ don't USE_COLORKEY
; #ACTORBASE=nMSPT // Basic formula for all the actors
; #ACTOR=id // Overridden formula for special cases
; {
; // nMSPT is Milliseconds per Tick
; // Behaviors
; 0=nRepeatCount, (cell index(, delay ticks(, displacement_x,
displacement_y(, sound id)))...;
; // nRepeatCount = 0 for infinite loop, - value for pendulum movement
; // delay ticks; 5 = fast; 10 = moderate; 15 = slow
; }
; #COACTOR=...

#ACTORBASE=
{
    STANDF          =1, (0);
    STANDB          =1, (8);
    ...
}

#ACTOR=aman_000;
{
    SPECIAL          =1, ...
    ...
}
=====*/

#include "stdafx.h"

#include "ResMan.h"
#include "Parser.h"

#include "UC2Ani/DIB.h"
#include "UC2Ani/PhSprite.h"
#include "Behavior.h"
#include "Actor.h"
#include "UC2Ani/Sound.h"
#include "UC2Ani/MCIObj.h"

const CString SERVER_IP      = _T("206.111.133.50"); // Ftp Server IP
// This must be changed according to the Server path
const CString DYNAMIC_BANNER_PATH =
_T("www.unichat\\dynamicAdds\\Update.txt");

//const for Dyn_ads Path
const CString C_STR_FTP_DYNAMIC_ADS_PATH = _T("web/dyn_ads");

const CString C_STR_DYNAMIC_BANNER_INI_FILE = _T("Dybanner.ini");

```

```

extern CParser gParser;

double GetMapEditorVersion()
{
    // V0.90 Feb 1998
    // V0.99 Mar 7, 1998    integer Resource ID
    // V1.00 Mar 11, 1998   MIT merged into SIT, Hyperlink
    // V1.01 Mar 17, 1998   DEFAULT_IO=0x1000 => 0x0000
    return 1.01;
}

LPCTSTR RESFILE_FILTER = "Resource Information Tables(*.RIT)|*.RIT|"
                          "All files (*.*)|*.*|";
LPCTSTR RES_BMPEXT = ".bmp"; // ".bm"
LPCTSTR RES_JPTEXT = ".gif";
LPCTSTR ACTOR_SHADOW_NAME = "cs00|cshad001";
LPCTSTR PUBLIC_STAGE_NAME_FORMAT = "[p2/%s]";
LPCTSTR MUD_STAGE_NAME_FORMAT = "[u2/%s]00";
// SATYA CHANGES START
LPCTSTR HOME_STAGE_NAME_FORMAT = "[home/%s]";
const char UCServerIP[] = "206.111.133.50";
// SATYA CHANGES END

static const PALETTEENTRY apeMASTER[256] =
{
#include "U2Pal.Inc" // 256 Color Table
};

const char* DATA_PATH =
#ifdef _DEBUG
#ifdef MAPEDITOR
    "\\data\\";
#else
#ifdef _MALL
    "\\Mall\\";
#else
    "\\MapEd\\data\\";
#endif
#endif
#else
    "\\";
#endif

/////////////////////////////////////////////////////////////////
// SATYA CHANGES START
// Implementation for class CChnCatTree
CChnCatTree::CChnCatTree()
{
    m_strCatName = _T("");
    m_nNum = 0;
    m_pItem = NULL;
}

CChnCatTree::~CChnCatTree()
{
    if (m_pItem)

```



```

        delete [] m_pItem;
        m_pItem = NULL;

    }
    // SATYA CHANGES END

    //////////////////////////////////////

WORD CResMan::CSpriteInfo::GetType()
{
    int iDS = m_strName.Find('|');
    if (iDS >= 0)        // found
        iDS++;
    else
        iDS = 0;
    switch (tolower(m_strName[iDS]))
    {
        case 't':    return SPRITE_TILE;
        case 'w':    return SPRITE_WALL;
        case 'c':    return SPRITE_STATIC;
        case 'i':    return SPRITE_PHASED;
        case 'a':    return SPRITE_ACTOR;
    }
    return -1;
}

    //////////////////////////////////////

CResMan::CResMan()
{
    // SATYA CHANGES START
    m_nChnCatTree      = 0;
    m_pChnCatTree      = NULL;
    m_pInetSes         = NULL;
    m_pFtpCon           = NULL;
    m_iMainCat          = 0;
    m_iSubCat           = 0;
    // SATYA CHANGES END

    TRACE0("CResMan::CResMan()\n");
    m_nSprites          = 0;
    m_aSI               = NULL;
    m_nWaves            = 0;
    m_aWave             = NULL;
    m_nActorDescs       = 0;
    m_aActorDesc        = NULL;
    m_nStageNames       = 0;
    m_aStageName        = NULL;
    m_nServerIPs        = 0;
    m_aServerIP         = NULL;
    m_nResFiles         = 1;
    m_strResFile.Empty();
    m_bMute              = FALSE;
    m_nBubbleTextLimit  = GetDefaultBubbleTextLimit();
    m_nBubbleTime       = GetDefaultBubbleTime();

```

```

    CString strRes;
    int len=512;
    char* psz = strRes.GetBuffer(len);
    ::GetCurrentDirectory(len, psz);
    strRes.ReleaseBuffer();
    strRes += DATA_PATH;
    SetResPath(strRes);
    TRACE("CResMan::CResMan - SetResPath(%s)\n", strRes);
}

CResMan::~CResMan()
{
    TRACE0("CResMan::~CResMan()\n");
    DeleteResources();
}

void CResMan::DeleteResources()
{
    if (m_aSI)
        delete [] m_aSI;
    m_aSI = NULL;
    m_nSprites = 0;

    if (m_aWave)
    {
        for (int i=0; i < m_nWaves; i++)
        {
            CWaveData& wd = m_aWave[i];
            if (wd.m_pWave)
                delete wd.m_pWave;
        }
        delete [] m_aWave;
    }
    m_aWave = NULL;
    m_nWaves = 0;
    if (m_aMIDI)
        delete [] m_aMIDI;
    m_nMIDIs = 0;

    if (m_aActorDesc)
        delete [] m_aActorDesc;
    m_aActorDesc = NULL;
    m_nActorDescs = 0;

    if (m_aStageName)
        delete [] m_aStageName;
    m_nStageNames = 0;

    if (m_aServerIP)
        delete [] m_aServerIP;
    m_nServerIPs = 0;

    RemoveAllDIBs();

    //CHANGES_MADE_FOR_UNICHAT_2

    //delete the FTP connection

```

```

    if(m_pFtpCon != NULL)
    {
        delete m_pFtpCon;
        m_pFtpCon = NULL;
    }

    //delete internet session.
    if(m_pInetSes)
    {
        delete m_pInetSes;
        m_pInetSes = NULL;
    }
}

void CResMan::CountItems(CTextFileBuffer& tfb)
{
    m_nSprites = 0;
    m_nWaves = 0;
    m_nMIDIs = 0;
    m_nStageNames = 0;
    m_nServerIPs = 0;
    int nAD=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine()) // At first, check if it's a
comment line
            continue;
        CString strBuf;
        if (!gParser.SetLeftToken('#') ||
            !gParser.GetValueRightToken(strBuf, '='))
            continue; // get next line
        if ((lstrcmpi(strBuf, "TIL") == 0) || // matching!
            (lstrcmpi(strBuf, "STT") == 0) ||
            (lstrcmpi(strBuf, "ANI") == 0) ||
            (lstrcmpi(strBuf, "AVT") == 0))
        {
            m_nSprites += CountContents(tfb);
        }
        else if (lstrcmpi(strBuf, "WAV") == 0)
        {
            m_nWaves += CountContents(tfb, ';');
        }
        else if (lstrcmpi(strBuf, "MID") == 0)
        {
            m_nMIDIs += CountContents(tfb, ';');
        }
        else if (lstrcmpi(strBuf, "STAGE") == 0)
        {
            m_nStageNames += CountContents(tfb, ';');
        }
        else if (lstrcmpi(strBuf, "SERVERIP") == 0)
        {
            m_nServerIPs += CountContents(tfb, ';');
        }
        else if (lstrcmpi(strBuf, "ACTOR") == 0) // matching!
        {

```

```

        m_aActorDesc[nAD++] .SetNumBehaviors(CountContents(tfb));
    }
    tfb.SeekToBegin();
}

// Count the number of lines that matches a specified string between '#' and
// '='
int CResMan::CountHeaders(CTextFileBuffer& tfb, const char* szToken)
{
    int i=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine()) // At first, check if it's a
comment line
            continue;
        CString strBuf;
        if (!gParser.SetLeftToken('#') || // get next line
            !gParser.GetValueRightToken(strBuf, '='))
            continue;
        if (lstrcmpi(strBuf, szToken) == 0) // matching!
            i++;
    }
    tfb.SeekToBegin();
    return i;
}

// Count the number of lines that include token('=') between { and }
int CResMan::CountContents(CTextFileBuffer& tfb, const char token)
{
    int i=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine()) // At first, check if it's a
comment line
            continue;
        if (gParser.SetLeftToken('{') // Begin
            {
                if (gParser.SetLeftToken('}') // {}
                    break;
                continue; // get next line
            }
        if (gParser.SetLeftToken('}') // End of contents
            break; // out of while loop
        if (gParser.SetLeftToken(token))
            i++; // Anyway, we should increase the counter
    }
    return i;
}

BOOL CResMan::Initialize(CTextFileBuffer& tfb)
{
    DeleteResources();

    m_nActorDescs = CountHeaders(tfb, "ACTOR");
}

```

```

////////// Initialize Actor Behaviors
if (m_nActorDescs <= 0)
{
    AfxMessageBox(IDS_RIT_NO_BEHAVIORS);
//    return FALSE;
}
else
{
    m_aActorDesc = new CActorDesc[m_nActorDescs];
}
TRACE("# of Actor Descriptions = %d\n", m_nActorDescs);

CountItems(tfb);

////////// Initialize Sprite Informations
if (m_nSprites <= 0)
{
    AfxMessageBox(IDS_RIT_NO_SPRITES);
    return FALSE;
}
else
{
    m_aSI = new CSpriteInfo[m_nSprites];
}
TRACE("# of Sprites = %d\n", m_nSprites);

if (m_nWaves <= 0)
{
//    AfxMessageBox(IDS_RIT_NO_WAVES);
//    return FALSE;
}
else
{
    m_aWave = new CWaveData[m_nWaves];
}
TRACE("# of Waves = %d\n", m_nWaves);

if (m_nMIDIs <= 0)
{
//    AfxMessageBox(IDS_RIT_NO_MIDIS);
//    return FALSE;
}
else
{
    m_aMIDI = new CString[m_nMIDIs];
}
TRACE("# of MIDIs = %d\n", m_nMIDIs);

if (m_nStageNames <= 0)
{
//    AfxMessageBox(IDS_RIT_NO_STAGES);
//    return FALSE;
}
else
{
    m_aStageName = new CString[m_nStageNames];
}

```

```

TRACE("# of Stage Names = %d\n", m_nStageNames);

if (m_nServerIPs <= 0)
{
    AfxMessageBox(IDS_RIT_NO_SERVERIPS);
//    return FALSE;
}
else
{
    m_aServerIP = new CString[m_nServerIPs];
}
TRACE("# of Server IPs = %d\n", m_nServerIPs);

return TRUE;
}

BOOL CResMan::SetResPath(LPCSTR path)
{
    m_strResPath = path;
    int len = lstrlen(path);
    if (path[len-1] != '\\')
        m_strResPath += "\\";
    return TRUE;
}

// Modify this so that it can handle multiple RIT files (resrc00?.rit)
BOOL CResMan::Load(LPCSTR path)
{
    TRACE0("CResMan::Load()\n");
    //CHANGES_MADE_FOR_UNICHAT_2
    // Down load the Dynamic Banner Info
    if (!DownloadDynamicBannerInfo())
    {
        return FALSE;
    }

    if ((path == NULL) || (lstrlen(path) == 0))
    {
        ::SetCurrentDirectory(m_strResPath);
        // Show an Open File dialog to get the name.
        CFileDialog dlg(TRUE, // Open
                        NULL, // No default extension
                        m_strResFile, // Initial file name
                        OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
                        RESFILE_FILTER);

        if (dlg.DoModal() == IDOK)
            m_strResFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        m_strResFile = path;
        IncludePath(m_strResFile);
    }
}

```

```

CTextFileBuffer tfb(gParser.GetMaxBuffer());
if (!tfb.Load(m_strResFile))
{
    return FALSE;
}

ExcludePath(m_strResFile);

if (!Initialize(tfb))
{
    return FALSE;
}

int i=0;
int ad=0;
CString strTemp;
BOOL bOldVersion = FALSE;

while (tfb.ReadString())
{
    gParser.CopyBuffer(tfb.GetString());
    if (gParser.IsCommentLine())
        continue;
    if (!gParser.SetLeftToken('#'))
        continue;
    CString strBuf;
    if (!gParser.GetValueRightToken(strBuf, '='))
        continue;

    //////////////////////////////////////
    if (lstrcmpi(strBuf, "VERSION") == 0)
    {
        gParser.SetLeftToken('=');
        double fVersion;
        gParser.GetValueRightToken(fVersion, ';');
        if (fVersion <= 0.90)
        {
            CString strMsg;
            strMsg.Format("Old Version %.2f", fVersion);
            AfxMessageBox(strMsg);
            bOldVersion = TRUE;
        }
    }
    else if (lstrcmpi(strBuf, "TIL") == 0) // matching!
    {
        while (tfb.ReadString())
        {
            gParser.CopyBuffer(tfb.GetString());
            if (gParser.IsCommentLine())
                continue;
            if (gParser.SetLeftToken('{'))
            {
                if (gParser.SetLeftToken('}'))
                    break;
                continue;
            }
            if (gParser.SetLeftToken('}'))

```

```

        break;
        if (!gParser.GetValueRightToken(strTemp, '='))
            continue;
        m_aSI[i].m_strName = strTemp;
        gParser.GetValueRightToken(m_aSI[i].m_sColRow, ',',
';');

        m_aSI[i].m_sEarth = CSize(64/2,0);
        gParser.GetValueRightToken(m_aSI[i].m_strHyperlink,
',', ';');

        i++;
    }
}
else if (lstrcmpi(strBuf, "STT") == 0)
{
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, '='))
            continue;
        strTemp.MakeLower();
        m_aSI[i].m_strName = strTemp;
        m_aSI[i].m_sColRow = CSize(1, 1);
        gParser.GetValueRightToken(m_aSI[i].m_sEarth, ',',
';');

        gParser.GetValueRightToken(m_aSI[i].m_strHyperlink,
',', ';');

        i++;
    }
}
else if (lstrcmpi(strBuf, "ANI") == 0)
{
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, '='))
            continue;
        strTemp.MakeLower();

```



```

        m_aSI[i].m_strName = strTemp;
        gParser.GetValueRightToken(m_aSI[i].m_sColRow, ',');
        gParser.GetValueRightToken(m_aSI[i].m_sEarth, ',',
';');

        gParser.GetValueRightToken(m_aSI[i].m_strHyperlink,
',', ';');

        i++;
    }
}
else if (lstrcmpi(strBuf, "AVT") == 0)
{
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, '='))
            continue;
        strTemp.MakeLower();
        m_aSI[i].m_strName = strTemp;
        gParser.GetValueRightToken(m_aSI[i].m_sColRow, ',');
        gParser.GetValueRightToken(m_aSI[i].m_sEarth, ',',
';');

        gParser.GetValueRightToken(m_aSI[i].m_strHyperlink,
',', ';');

        i++;
    }
}
else if (lstrcmpi(strBuf, "WAV") == 0)
{
    int w=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, ';'))
            continue;
        if (w >= m_nWaves)
            break;
        strTemp.MakeLower();

```

```

        m_aWave[w++].m_strName = strTemp;
    }
}
else if (lstrcmpi(strBuf, "MID") == 0)
{
    int m=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, ';'))
            continue;
        if (m >= m_nMIDIs)
            break;
        strTemp.MakeLower();
        m_aMIDI[m++] = strTemp;
    }
}
else if (lstrcmpi(strBuf, "STAGE") == 0)
{
    int s=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, ';'))
            continue;
        if (s >= m_nStageNames)
            break;
        strTemp.MakeLower();
        m_aStageName[s++] = strTemp;
    }
}
else if (lstrcmpi(strBuf, "SERVERIP") == 0)
{
    int p=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());

```

```

        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, ';'))
            continue;
        if (p >= m_nServerIPs)
            break;
        m_aServerIP[p++] = strTemp;
    }
}
else if (lstrcmpi(strBuf, "ACTOR") == 0)
{
    if (ad >= m_nActorDescs)
        break;
    m_aActorDesc[ad++].Load(tfb);
}
else
{
    TRACE("Unknown Data type in RIT file(%s)!\n",
m_strResFile);
}
}
/*
#ifdef _DEBUG
    for (i=0; i < m_nSprites; i++)
        TRACE("%d)\t%s=(%d,%d), (%d,%d)\n", i, m_aSI[i].m_strName,
m_aSI[i].m_sColRow.cx, m_aSI[i].m_sColRow.cy,
        m_aSI[i].m_sEarth.cx, m_aSI[i].m_sEarth.cy);
    for (i=0; i < ad; i++)
        TRACE("%d)\t%d behaviors\n", i,
m_aActorDesc[i].GetNumBehaviors());
#endif
*/

for (int s=0; s < GetNumStageNames(); s++)
{
    CString strStageID(m_aStageName[s]);
    CString strTitle(strStageID);
    if (GetStageTitle(strTitle) // access each sit file
    {
        MakeStageName(strStageID, TRUE);
        m_aStageName[s] = strStageID + ' ' + strTitle;
    }
    else
    {
        TRACE("SIT(%s) file not found!\n", strTemp);
    }
}

LoadWave();

```

```

    // SATYA CHANGES START
    // load the room catigories
    LoadChnCatTree();
    // SATYA CHANGES END

    return TRUE;
}

int CResMan::GetSpriteID(const CString& strName) const
{
    for (int i=0; i < m_nSprites; i++)
    {
        if (lstrcmpi(m_aSI[i].m_strName, strName) == 0)
            return i;
    }
    return -1; // Not Found!
}

int CResMan::GetWaveID(const CString& strName) const
{
    for (int i=0; i < m_nWaves; i++)
    {
        if (lstrcmpi(m_aWave[i].m_strName, strName) == 0)
            return i;
    }
    return -1; // Not Found!
}

int CResMan::GetMIDIID(const CString& strName) const
{
    for (int i=0; i < m_nMIDIs; i++)
    {
        if (lstrcmpi(m_aMIDI[i], strName) == 0)
            return i;
    }
    return -1; // Not Found!
}

BOOL CResMan::LoadWave()
{
    CString strFile;
    for (int i=0; i < m_nWaves; i++)
    {
        CWaveData& wd = m_aWave[i];
        ASSERT(!wd.m_strName.IsEmpty());
        if (wd.m_strName[0] == 's') // wave
        {
            wd.m_pWave = new CSound;
            strFile = m_strResPath + wd.m_strName + ".wav";
            TRACE("Loading Wave %s\n", strFile);
            if (!wd.m_pWave->Load(strFile))
            {
                delete wd.m_pWave;
                wd.m_pWave = NULL;
            }
        }
    }
}

```

```

        }
    }
    else
    {
        wd.m_pWave = NULL;
    }
}
return TRUE;
}

// Play wave sound (memory-loaded)
BOOL CResMan::PlayWave(const int nID) const
{
    if (m_bMute)
        return FALSE;
    CWaveData& wd = m_aWave[nID];
    CString& s = wd.m_strName;

    if (s[0] == 's') // wave
    {
        if (!wd.m_pWave)
            return FALSE;
        wd.m_pWave->Play();
        return TRUE;
    }
    else
    {
        TRACE("Not wave sound\n");
        return FALSE;
    }
    return FALSE;
}

WORD CResMan::GetSpriteTypeByName(const CString& strName) const
{
    CString strRes(strName);
    MakeResName(strRes);
    int i = GetSpriteID(strRes);
    return (i >= 0) ? GetSpriteType(i) : SPRITE_STATIC;
}

////////////////////////////////////
// DIB Management
CDIB* CResMan::LoadDIB(LPCSTR szResName)
{
    CString strFile(szResName);
    ExpandResName(strFile, RES_BMPEXT);
    strFile.MakeLower();
    CDIB* pDIB;
    POSITION pos = m_oldDIB.GetHeadPosition();
    while (pos)
    {
        pDIB = (CDIB*)m_oldDIB.GetNext(pos); // Increment position.
        if (pDIB)
        {
            ASSERT(pDIB->IsKindOf(RUNTIME_CLASS(CDIB)));
            CString* pstr = pDIB->GetName();

```

```

        if (*pstr == strFile)    // compare with only filename
(excluding path)
            return pDIB;
    }
}
// File not found in current resource bank.
// So create a new one and add it to the bank.
pDIB = new CDIB;
if (!pDIB->Load(strFile))
{
    TRACE("%s: Resource Load Failure!\n", strFile);
    delete pDIB;
    return NULL;
}
LoadMasterPalette(pDIB);
m_oldDIB.AddTail(pDIB);
return pDIB;
}

CDIB* CResMan::LoadDIB(const WORD wResid, const BOOL bMasterPalette)
{
    CString strName;
    strName.Format("RES:%d", wResid);
    CDIB* pDIB;
    POSITION pos = m_oldDIB.GetHeadPosition();
    while (pos)
    {
        pDIB = (CDIB*)m_oldDIB.GetNext(pos); // Increment position.
        if (pDIB)
        {
            ASSERT(pDIB->IsKindOf(RUNTIME_CLASS(CDIB)));
            CString* pstr = pDIB->GetName();
            if (*pstr == strName)
                return pDIB;
        }
    }
    // File not found in current resource bank.
    // So create a new one and add it to the bank.
    pDIB = new CDIB;
    if (!pDIB->Load(wResid))
    {
        TRACE("%d: Resource Load Failure!\n", wResid);
        delete pDIB;
        return NULL;
    }
    if (bMasterPalette)
        LoadMasterPalette(pDIB);
    m_oldDIB.AddTail(pDIB);
    return pDIB;
}

// Load the specified resource into CPhasedSprite
// and sets initial conditions such as # of cells.
// The caller is responsible for deleting this sprite.
CPhasedSprite* CResMan::LoadPhasedSprite(LPCSTR szResName, const BOOL
bDIBReuse, int nResID)
{

```

```

int nResID;

if (IsDynamicBanner(nResID))
{
    nResID = nResID;
}
else
{
    nResID = GetSpriteID(szResName);
}
if (nResID < 0)    // Not found in RIT
{
    CString strMsg;
    strMsg.Format("%s not found in RIT", szResName);
    AfxMessageBox(strMsg);
    return NULL;
}

CPhasedSprite* pPS;
if (bDIBReuse)    // Reuse Mechanism
{
    CDIB* pDIB = LoadDIB(szResName);
    if (!pDIB)
        return NULL;
    pPS = new CPhasedSprite;
    pPS->SetDIB(pDIB);    // link DIB resource to this sprite
}
else // Do not Reuse
{
    CString strFile(szResName);
    ExpandResName(strFile, RES_BMPEXT);

    pPS = new CPhasedSprite;
    if (!pPS->Load(strFile))
    {
        delete pPS;
        TRACE("%s: Resource Load Failure!\n", strFile);
        return NULL;
    }
    ASSERT(pPS->GetDIB());
    LoadMasterPalette(pPS->GetDIB());
#ifdef MAPEDITOR
    if (GetSpriteTypeByName(szResName) == SPRITE_ACTOR)
    {
        static int nColorSet = 1;
        RotateActorColorSet(pPS->GetDIB(), nColorSet);
        nColorSet++;
        if (nColorSet > 4)
            nColorSet = 1;
    }
#endif
}

pPS->SetType(GetSpriteType(nResID));
pPS->SetNumCells(GetSpriteColRow(nResID));
pPS->SetEarth(GetSpriteEarth(nResID));
CString* pStr = GetSpriteHyperlink(nResID);
if (!pStr->IsEmpty())

```

```

        pPS->SetHyperlink(*pStr);
    return pPS;
}

CActor* CResMan::LoadActor(const int nCharID, const int nColorSet, const BOOL
bME)
{
    CActorDesc* pAD = GetActorDesc(nCharID);
    if (!pAD)
    {
        CString strMsg;
        strMsg.Format("Actor Description: %d not found!", nCharID);
        AfxMessageBox(strMsg);
        return NULL;
    }

    CString strFile(*pAD->GetResName());
    ExpandResName(strFile, RES_BMPEXT);
    CActor* pA = new CActor;
    if (!pA->Load(strFile))
    {
        delete pA;
        TRACE("%s: Resource Load Failure!\n", strFile);
        return NULL;
    }
    ASSERT(pA->GetDIB());
    LoadMasterPalette(pA->GetDIB());

    if (bME)
        ShowOutline(pA->GetDIB());
    if (nColorSet)
        RotateActorColorSet(pA->GetDIB(), nColorSet);

    // Load Shadow
    CDIB* pDIB = LoadDIB(ACTOR_SHADOW_NAME); // LoadMasterPalette();

    if (pDIB)
    {
        CSprite* pS = new CSprite;
        pS->SetDIB(pDIB); // link DIB resource to this sprite
        pS->SetType(SPRITE_STATIC);
        int nResID = GetSpriteID(ACTOR_SHADOW_NAME);
        pS->SetEarth(GetSpriteEarth(nResID));
        pA->SetShadow(pS);
    }

    int nResID = GetSpriteID(*pAD->GetResName());
    ASSERT(nResID >= 0);
    pA->SetType(GetSpriteType(nResID), SPRITE_ANI_ACTOR);
    pA->SetNumCells(GetSpriteColRow(nResID));
    pA->SetEarth(GetSpriteEarth(nResID));
    pA->SetCharID(nCharID);
    CString* pStr = GetSpriteHyperlink(nResID);
    if (!pStr->IsEmpty())
        pA->SetHyperlink(*pStr);

    return pA;
}

```



```

}

CBehavior* CResMan::GetActorBehavior(const WORD nCharID, const int bi)
{
    CActorDesc* pAD = GetActorDesc(nCharID);
    if (!pAD)
    {
        AfxMessageBox("Actor not found!");
        return NULL;
    }
    CBehavior* pBeh = pAD->GetBehavior(bi);
    if (!pBeh) // If not found in this ActorDesc, use that of the standard
instead
    {
        pAD = GetActorDesc(0); // Standard ActorDesc
        ASSERT(pAD);
        return pAD->GetBehavior(bi); // return that of the standard
    }
    return pBeh;
}

void CResMan::DeletedIB(CDIB* pDIB)
{
    POSITION pos = m_oldDIB.Find(pDIB);
    if (pos) // Hunt for pDIB
    {
        m_oldDIB.RemoveAt(pos);
        delete pDIB;
    }
}

// Be sure not to do use DIBs allocated here elsewhere!!!
void CResMan::RemoveAllDIBs()
{
    // Walk down the list deleting objects as we go.
    // We need to do this here because the base class simply deletes the
pointers.
    POSITION pos = m_oldDIB.GetHeadPosition();
    CDIB* pDIB;
    while (pos)
    {
        pDIB = (CDIB*)m_oldDIB.GetNext(pos); // Increment position.
        if (pDIB)
        {
            ASSERT(pDIB->IsKindOf(RUNTIME_CLASS(CDIB)));
            delete pDIB;
        }
    }
    // Now call the base class to remove the pointers.
    m_oldDIB.RemoveAll();
}

//////////////////////////////////// Naming Handling Methods
// Extract filename only
// C:\UC2\Data\t00\tcity000.bmp ==> t00\tcity000
void CResMan::MakeResName(CString& strName) const
{

```

```

    int i = strName.ReverseFind('\\');
    int j = strName.ReverseFind('.');
    if ((i < 0) && (j < 0))
        return;

    char* p = strName.GetBuffer(256);
    if (j > 0)
        p[j] = NULL;          // Exclude file extension
    if (i >= 0)
        p += (i+1);
    CString strTemp((char*)p);
    strName.ReleaseBuffer();
    strName = strTemp;
    strName.MakeLower();
}

void CResMan::ExcludePath(CString& strName) const
{
    int i = strName.ReverseFind('\\');
    if (i < 0)
        return;

    char* p = strName.GetBuffer(256);
    p += (i+1);
    CString strTemp((char*)p);
    strName.ReleaseBuffer();
    strName = strTemp;
}

void CResMan::IncludePath(CString& strName) const
{
    if (strName.Find('\\') >= 0)
        return;
    CString strFN(m_strResPath);
    strFN += strName;
    strName = strFN;
}

// "t00|tcity000" => "C:\UC2\Data\t00|tcity000.bmp"
void CResMan::ExpandResName(CString& strName, LPCSTR szExt) const
{
    MakeResName(strName);
    CString strFN(m_strResPath);
    strFN += strName;
    if (szExt)
        strFN += szExt;
    strName = strFN;
}

// "[p2/0010csin]Castle of the Wind" => "0010csin"
BOOL CResMan::ExtractStageID(CString& strName) const
{
    if (strName.GetLength() <= 5)
        return FALSE;

    if ((strName[0] == '[') && (strName[2] == '2') && (strName[3] == '/'))
    {
        int i = strName.Find(']');
    }
}

```

```

        if (i < 4)
            return FALSE;
        CString strRes(strName.Mid(4, i-3-1));
        strName = strRes;
        return TRUE;
    }
    return FALSE;
}

// "0010csin" => "[p2/0010csin]"
// Public or MUD
void CResMan::MakeStageName(CString& strName, const BOOL bPublic, const BOOL
bHome) const
{
    CString strID(strName);
    strID.Format(bPublic ? PUBLIC_STAGE_NAME_FORMAT :
MUD_STAGE_NAME_FORMAT,
                strName);
    strName = strID;

    /*CString strID(strName);

    if (!bHome) {
        strID.Format(bPublic ? PUBLIC_STAGE_NAME_FORMAT :
MUD_STAGE_NAME_FORMAT, // "[ep0/%s]"
                    strName);
    }
    else {
        strID.Format(HOME_STAGE_NAME_FORMAT, strName);
    }
    strName = strID;*/

}

////////////////////// Palette Control
// Be sure to call these methods before calling MapColorsToPalette
UINT CResMan::LoadMasterPalette(CDIB* pDIB) const
{
    return pDIB->SetPaletteEntries(0, 256, (const
LPPALETTEENTRY)apeMASTER);
}

void CResMan::ShowOutline(CDIB* pDIB) const
{
    // Write Outline Color to Outline (Off) Index
    // pDIB->SetPaletteEntries(237, 1, (const
LPPALETTEENTRY)&(apeMASTER[240]));
    pDIB->CopyPaletteEntry(237, 240);
}

// nColorSet = 0,1,2,3
void CResMan::RotateActorColorSet(CDIB* pDIB, const int nColorSet) const
{
    static int COLORS = 16;
    static int COLORSET[] =
    {

```

```

    0,1, 0,2, 0,3,    // nColorSet=4,5,6,
    1,0, 1,2, 1,3,    // 7,8,9,
    2,0, 2,1, 2,3,    // 10,11,12,
    3,0, 3,1, 3,2,    // 13,14,15
};
int nCS = (nColorSet < COLORS) ? nColorSet : nColorSet % COLORS;
int nC = (nCS < 4) ? nCS : COLORSET[(nCS-4)*2];
int nH = (nCS < 4) ? nCS : COLORSET[(nCS-4)*2+1];
pDIB->RotatePaletteIndex(200, COLORS, nC*4);    // Clothes
pDIB->RotatePaletteIndex(220, COLORS, nH*4);    // Hair
}

/*
t00.uds
=====
t*.bmp

ca00.uds
=====
ca*.bmp ~ cc*.bmp

cd00.uds
=====
cd*.bmp ~ cf*.bmp

cg00.uds
=====
cg*.bmp ~ cp*.bmp

cs00.uds
=====
cs*.bmp ~ ct*.bmp

cw00.uds
=====
cw*.bmp ~
i*.bmp
w*.bmp
*/
// Add prefix for UDS resource name
void CResMan::PrefixUDS00(CString& strName) const
{
    int iDS = strName.Find('|');
    if (iDS >= 0)    // Found, no need to ...
        return;
    CString strDS;
    switch (tolower(strName[0]))
    {
    case 't':    strDS = "t";    break;
    case 'c':
        {
            char ch = tolower(strName[1]);
            if (ch <= 'c')    strDS = "ca";
            else if (ch <= 'f')    strDS = "cd";
            else if (ch <= 'p')    strDS = "cg";
            else if (ch <= 't')    strDS = "cs";
            else    strDS = "cw";
        }
    }
}

```

```

        break;
    }
    case 'i':
    case 'w':    strDS = "cw";        break;
    case 'a':    strDS = "a";        break;
    }
    strDS += "00|";
    strName = strDS + strName;
}

int CResMan::GetStageType(LPCTSTR szName) const
{
    if (!szName)
        return ST_INVALID;
    if ((lstrlen(szName) < 5) || (szName[0] != '['))
        return ST_OLD;
    if ((szName[2] == '2') && (szName[3] == '/'))
    {
        if (szName[1] == 'u')
            return ST_MUD;
        if (szName[1] == 'p')
            return ST_PUBLIC;
    }
    return ST_OLD;
}

// strTitle(IN: stage filename, OUT: stage title)
BOOL CResMan::GetStageTitle(CString& strTitle) const
{
    CString strFile(strTitle);
    ExtractStageID(strFile);
    if (strFile.IsEmpty())
        return FALSE;
    ExpandResName(strFile, ".sit");
    CTextFileBuffer tfb(gParser.GetMaxBuffer());
    if (!tfb.Load(strFile))
    {
        return FALSE;
    }

    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (!gParser.SetLeftToken('#'))
            continue;
        CString strBuf;
        if (!gParser.GetValueRightToken(strBuf, '='))
            continue;

        if (lstrcmpi(strBuf, "STAGE") == 0)
        {
            gParser.SetLeftToken('=');
            gParser.GetValueRightToken(strTitle, ',', ';');
            // remove [u2/0000abcd]
            int i = strTitle.Find('');

```

```

        if (i > 0)
        {
            int n = strTitle.GetLength() - (i+1);
            CString strTemp(strTitle);
            strTitle = strTemp.Right(n);
        }
        return TRUE;
    }
    return FALSE;
}

// SATYA CHANGES START

// Function Name : LoadChnCatTree
// Inputs : Path of the File
// Purpose : Load the Room categories from Specified file
// Return : TRUE if it reads Successfully other wise FALSE
BOOL CResMan::LoadChnCatTree(LPCSTR path)
{
    TRACE0("CResMan::LoadChanTree()\n");

    CString strChanTreeFile = path;
    IncludePath(strChanTreeFile);

    CTextFileBuffer tfb(gParser.GetMaxBuffer());

    if (!tfb.Load(strChanTreeFile))
    {
        return FALSE;
    }

    ExcludePath(strChanTreeFile);

    DeleteResourcesChnCatTree();

    tfb.SeekToBegin();
    int i=0; // number of root items
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine()) // At first, check if it's a
comment line
            continue;
        if (gParser.SetLeftToken('#'))
            i++;
    }

    m_nChnCatTree = i;
    m_pChnCatTree = new CChnCatTree[m_nChnCatTree];

    tfb.SeekToBegin();
    i=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())

```

```

        continue;
    if (!gParser.SetLeftToken('#') ||
        !gParser.GetValueRightToken(m_pChnCatTree[i].m_strCatName,
        '='))
        continue;
    if (!gParser.GetValueRightToken(m_pChnCatTree[i].m_nNum, ';'))
        continue;

    m_pChnCatTree[i].m_pItem = new
CChnCatTreeItem[m_pChnCatTree[i].m_nNum];
    int j=0;
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if
(!gParser.GetValueRightToken(m_pChnCatTree[i].m_pItem[j].m_strItemName, ';'))
            continue;
        m_pChnCatTree[i].m_pItem[j].m_ItemID.cx = i;
        m_pChnCatTree[i].m_pItem[j].m_ItemID.cy = j;
        j++;
    }
    i++;
    //TRACE("# of Ads = %d\n", m_nSpritesD);
    return TRUE;
}
// Function Name : DeleteResourcesChnCatTree
// Inputs : none
// Purpose : Delete the Room catigories from the memory
// Return : void

void CResMan::DeleteResourcesChnCatTree()
{
    if (m_pChnCatTree)
        delete [] m_pChnCatTree;
    m_pChnCatTree = NULL;
    m_nChnCatTree = 0;
}
// Function Name : GetNumSubItemChanTree
// Inputs : Index (int) Index of the main Item
// Purpose : It finds the total number of SubItems under that main Item
// Return : int Number Sub items

int CResMan::GetNumSubItemChanTree(const int nIndex) const
{
    if (nIndex >= m_nChnCatTree)
        return -1;

```

```

        return m_pChnCatTree[nIndex].m_nNum;
    }

// Function Name : GetNameSubItemChanTree
// Inputs       : (int) cx -- Index of main Item (int) cy -index Subitem
// Purpose      : It return the name of th Item
// Return       : Name (CString) -- name of the Item

CString CResMan::GetNameSubItemChanTree(const int cx, const int cy) const
{
    if (cx >= m_nChnCatTree)
        return _T("");
    if (cy >= m_pChnCatTree[cx].m_nNum)
        return _T("");
    return m_pChnCatTree[cx].m_pItem[cy].m_strItemName;
}

// Function Name : GetNameSubItemChanTree
// Inputs       : (int) cx -- Index of main Item (int) cy -index Subitem
// Purpose      : It return the name of th Item
// Return       : Name (CString) -- name of the Item

CString CResMan::GetNameChanTree(const int nIndex) const
{
    if (nIndex >= m_nChnCatTree)
        return _T("");
    return m_pChnCatTree[nIndex].m_strCatName;
}

// Function Name : GetLatestDynamicBanner
// Inputs       : Name for the Dynamic banner
// Purpose      : It finds the name of the Dynamic Banner
// Return       : Name of the Dynamic Banner
//CHANGES_MADE_FOR_UNICHAT_2

BOOL CResMan::GetLatestDynamicBanner(CString &BannerPath)
{
    CString MainCat, SubCat;
    MainCat = GetNameChanTree(m_iMainCat);
    SubCat = GetNameSubItemChanTree(m_iMainCat, m_iSubCat);
    CString strBannerID = MainCat + "->" + SubCat;

    //m_strResPath
    CString strFileName =
    GetUnichatProfileString(m_strDynamicBannerLocalPath, strBannerID, "DYBANNERPATH
    ");
    if (strFileName.IsEmpty())
    {
        //unable to get the dynamic banner file
        return FALSE;
    }
    strFileName += _T(".bmp");
    CString strcurrFilePath = m_strResPath + strFileName;

    CString strModtime =
    GetUnichatProfileString(m_strDynamicBannerLocalPath, strBannerID, "LASTMODIFIED
    ");
    COleDateTime dtModRemoteBannerTime;
    dtModRemoteBannerTime.ParseDateTime(strModtime);

```



```

        BOOL bBannerDownLoad = FALSE;
        CFileFind FileFinder;
        if (FileFinder.FindFile(strcurrFilePath))
        {
            FileFinder.FindNextFile();
            FILETIME tBannerCreateTime;
            FileFinder.GetCreationTime(&tBannerCreateTime);
            COleDateTime dtLocalDyBannerTime(tBannerCreateTime);
            //if the remote time of the Dyanmic banner is greater than local
file timetime
            if(dtModRemoteBannerTime > dtLocalDyBannerTime)
            {
                bBannerDownLoad = TRUE;
            }
        }
        else
        {
            bBannerDownLoad = TRUE;
        }
        if (bBannerDownLoad)
        {
            BOOL bResult = FALSE;
            // Down load the from ftp file
            if(m_pInetSes == NULL)
            {
                m_pInetSes = new CInternetSession;
                ASSERT(m_pInetSes != NULL);
            }

            if(m_pFtpCon == NULL)
            {
                m_pFtpCon = m_pInetSes-
>GetFtpConnection(_T("www.unichat.com"),
_T("dev@unichat.com"),
_T("home1")

            );

                ASSERT(m_pFtpCon!=NULL);
            }

            //remote path only file name because the we are in correct dir.
            CString strRemotePath = C_STR_FTP_DYNAMIC_ADS_PATH +
_T("/") + strFileName;

            //now down laod the Dynamic banner file to local directory
            bResult = m_pFtpCon-
>GetFile(strRemotePath, strcurrFilePath, FALSE);
            if(bResult == FALSE)
            {
                return FALSE;
            }
        }
        BannerPath = strcurrFilePath ;
        return TRUE;

```

```

}

//CHANGES_MADE_FOR_UNICHAT_2

BOOL CResMan::AddDynamicBanner(CString BannerName)
{
    return TRUE;
}

// SATYA CHANGES END

void CResMan::SetRoomCatigories(int iMain, int iSub)
{
    m_iMainCat = iMain;
    m_iSubCat = iSub;
}

//CHANGES_MADE_FOR_UNICHAT_2
CString* CResMan:: GetSpriteHyperlink(const int nID)
{
    CString HyperLink;
    if (IsDynamicBanner(nID))
    {
        // Open the URL where we can the Url form the Server
        //m_pInetSes->OpenURL();
        // Read the Hyper link in to the String and pass
        // copy to the Sting and return..
        //return &HyperLink
        // return &(m_aSI[nID].m_strHyperlink);
        CString MainCat,SubCat;
        MainCat = GetNameChanTree(m_iMainCat);
        SubCat = GetNameSubItemChanTree(m_iMainCat,m_iSubCat);
        CString strBannerID = MainCat + "->" + SubCat;
        CString strURL =
        GetUnichatProfileString(m_strDynamicBannerLocalPath,strBannerID,"URL");
        m_aSI[nID].m_strHyperlink = strURL;
        return &(m_aSI[nID].m_strHyperlink);
    }
    else
    {
        return &(m_aSI[nID].m_strHyperlink);
    }
}

//CHANGES_MADE_FOR_UNICHAT_2
BOOL CResMan:: DownloadDynamicBannerInfo()
{
    // down load the Ini file
    try
    {
        if(m_pInetSes == NULL)
        {
            m_pInetSes = new CInternetSession;
            ASSERT(m_pInetSes != NULL);
        }
    }
}

```

```

        if(m_pFtpCon == NULL)
        {
            m_pFtpCon = m_pInetSes-
>GetFtpConnection(_T("www.unichat.com"),
_T("dev@unichat.com"),
_T("home1")
        );
            ASSERT(m_pFtpCon!=NULL);
        }

        BOOL bResult =m_pFtpCon-
>SetCurrentDirectory(C_STR_FTP_DYNAMIC_ADS_PATH);
        if(bResult == FALSE)
        {
            AfxMessageBox(_T("Unable change the directory on FTP
server."),MB_OK|MB_ICONSTOP);
            return FALSE;
        }

        //now down laod the .ini file to local directory
        m_strDynamicBannerLocalPath = m_strResPath +
C_STR_DYNAMIC_BANNER_INI_FILE;
        bResult = m_pFtpCon-
>GetFile(C_STR_DYNAMIC_BANNER_INI_FILE,m_strDynamicBannerLocalPath,FALSE);
        if(bResult == FALSE)
        {
            AfxMessageBox(_T("Unable get the file from on FTP
server."),MB_OK|MB_ICONSTOP);
            return FALSE;
        }
    }
    catch(CInternetException *pE)
    {
        TCHAR szBuffer[2048];
        memset(szBuffer,0,2048);
        pE->GetErrorMessage(szBuffer,2048);
        AfxMessageBox(szBuffer,MB_OK|MB_ICONSTOP);

        if(m_pInetSes)
        {
            delete m_pInetSes;
            m_pInetSes = NULL;
        }
        pE->Delete();
        return FALSE;
    }
    return TRUE;
}

//CHANGES_MADE_FOR_UNICHAT_2

```

```
CString CResMan:: GetUnichatProfileString(CString strIniFilePath,CString
strKeyName,CString strSectionName)
{
    TCHAR szBuffer[1024];
    memset(szBuffer,0,1024);

    GetPrivateProfileString(strKeyName,strSectionName,NULL,szBuffer,1024,st
rIniFilePath);
    CString strResultstrResult = szBuffer;
    return strResultstrResult;
}
```

ResMan.h

```
//      ResMan: Resource Manager
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT NETWORKS INC
//=====

#ifndef __RESMAN_H
#define __RESMAN_H
#include <afxinet.h>
#include "Behavior.h"
////////////////////////////////////
// #include "UCServerSession.h"
class CDIB;
class CPhasedSprite;
class CActor;
class CTextFileBuffer;
class CSound;

enum UC2CHANNEL_TYPE
{
    ST_INVALID,
    ST_OLD,           // Old version
    ST_MUD,           // MUD mode
    ST_PUBLIC        // User Creatable
};

// SATYA CCHANGES START
class CChnCatTreeItem
{
public:
    CSize m_ItemID;
    CString m_strItemName;
};

class CChnCatTree
{
public:
    CChnCatTree();
    virtual ~CChnCatTree();
    CString m_strCatName;
    int m_nNum;
    CChnCatTreeItem* m_pItem;
};
// SATYA CCHANGES END

class CResMan : public CObject
{
public:
    BOOL AddDynamicBanner(CString BannerName);
    CString GetUnichatProfileString(CString strIniFilePath, CString
strKeyName, CString strSectionName);
    CResMan();
    virtual ~CResMan();

    class CSpriteInfo
```

```

    {
    public:
        WORD          GetType();
        CString       m_strName; // should be an internal resource
ID
        CSize        m_sColRow; // # of Columns and Rows of the Cells
        CSize        m_sEarth;  // Center of Earth (offset)
        CString       m_strHyperlink;
    };

    class CWaveData
    {
    public:
        CString       m_strName; // resource name
        CSound*       m_pWave;
    };

// Attributes
    int              GetNumSprites() const { return m_nSprites; }

    int              GetSpriteID(const CString& strName) const;
    WORD            GetSpriteType(const int nID) const
    { return m_aSI[nID].GetType(); }
    // C++ cannot differentiate const int and const CString& ?
    WORD            GetSpriteTypeByName(const CString& strName) const;
    CString*        GetSpriteName(const int nID)
    { return ((nID >= 0) && (nID < m_nSprites)) ?
&(m_aSI[nID].m_strName) : NULL; }
    CSize          GetSpriteColRow(const int nID) const
    { return m_aSI[nID].m_sColRow; }
    CPoint          GetSpriteEarth(const int nID) const
    { return m_aSI[nID].m_sEarth; }
    CString*        GetFileName() { return &m_strResFile; }
    CString*        GetResPath() { return &m_strResPath; }
    int             GetNumResFiles() const { return m_nResFiles; }
    int             GetNumActorDescs() const { return m_nActorDescs; }
    CActorDesc*     GetActorDesc(const WORD nCharID)
    { return (nCharID < m_nActorDescs) ?
&m_aActorDesc[nCharID] : NULL; }
    CBehavior*      GetActorBehavior(const WORD nCharID, const int bi);

    int             GetWaveID(const CString& strName) const;
    int             GetMIDIID(const CString& strName) const;
    CString*        GetMIDIName(const int nID)
    { return (nID < m_nMIDIs) ? &m_aMIDI[nID] : NULL; }

    int             GetNumStageNames() const { return m_nStageNames; }
}
    CString*        GetStageName(const int nID)
    { return (nID < m_nStageNames) ? &m_aStageName[nID] :
NULL; }
    int             GetStageType(LPCTSTR szName) const;

    int             GetNumServerIPs() const { return m_nServerIPs; }
}
    CString*        GetServerIP(const int n)
    { return (n < m_nServerIPs) ? &m_aServerIP[n] : NULL; }
}

```

```

        int                GetBubbleTextLimit() const                { return
m_nBubbleTextLimit; }
#ifdef _MALL
        int                GetDefaultBubbleTextLimit() const        { return 255; }
#else
        int                GetDefaultBubbleTextLimit() const        { return 100; }
#endif
        int                GetBubbleTime() const                    { return
m_nBubbleTime; }
        int                GetDefaultBubbleTime() const            { return 4000; }

// Operations
void                DeleteResources();
BOOL                Load(LPCSTR path="u2res000.rit");
BOOL                SetResPath(LPCSTR path=NULL);
void                MakeResName(CString& strName) const;            // Extract
filename only
void                ExpandResName(CString& strName, LPCSTR szExt=NULL) const;
void                ExcludePath(CString& strName) const;
void                IncludePath(CString& strName) const;
BOOL                ExtractStageID(CString& strName) const;
void                MakeStageName(CString& strName, const BOOL bPublic,const
BOOL bHome=FALSE) const;
BOOL                GetStageTitle(CString& strTitle) const;

void                PrefixUDS00(CString& strName) const;

BOOL                PlayWave(const int nID) const;
void                SetMute(const BOOL bMute)            { m_bMute = bMute; }
// SATYA CHANGES START
void                DeleteResourcesChnCatTree();
BOOL                LoadChnCatTree(LPCSTR path="uc_chnl_cat.cpc");
int                GetNumChanTree() const {return m_nChnCatTree;};
CString            GetNameChanTree(const int nIndex) const;
int                GetNumSubItemChanTree(const int nIndex) const;
CString            GetNameSubItemChanTree(const int cx, const int cy)
const;
BOOL                GetLatestDynamicBanner(CString &BannerPath);
CString*            GetSpriteHyperlink(const int nResID);
// SATYA CHANGES END

// DIB Management
CDIB*                LoadDIB(LPCSTR szResName);
CDIB*                LoadDIB(const WORD wResid, const BOOL
bMasterPalette=FALSE);
CPhasedSprite*      LoadPhasedSprite(LPCSTR szResName, const BOOL
bDIBReuse=TRUE,int nResID = -1);
CActor*             LoadActor(const int nCharID, const int
nColorSet=0, const BOOL bME=FALSE);
void                DeleteDIB(CDIB* pDIB);
void                RemoveAllDIBs();
UINT                LoadMasterPalette(CDIB* pDIB) const;
void                ShowOutline(CDIB* pDIB) const;

```

```

        void                RotateActorColorSet(CDIB* pDIB, const int nColorSet)
const;
        int                SetBubbleTextLimit(const int nBTL) { return
m_nBubbleTextLimit = nBTL; }
        int                SetBubbleTime(const int nBT) { return
m_nBubbleTime = nBT; }

protected:
        BOOL LoadWave();
        int CountHeaders(CTextFileBuffer& tfb, const char* szToken);
        void CountItems(CTextFileBuffer& tfb);
        int CountContents(CTextFileBuffer& tfb, const char token=' ');
        BOOL Initialize(CTextFileBuffer& tfb);

        int                m_nSprites;
        CspriteInfo*       m_aSI;
        COBList             m_oldDIB;
        CActorDesc*        m_aActorDesc;
        int                m_nActorDescs;
        CString*           m_aStageName; // list of User-creatable stages'
file name
        int                m_nStageNames;
        CString*           m_aServerIP;
        int                m_nServerIPs;
        CWaveData*         m_aWave;
        int                m_nWaves;
        CString*           m_aMIDI;
        int                m_nMIDIs;
        CString            m_strResFile; // Resource Information:
filename.ext only
        CString            m_strResPath; // Resource Path: with
trailing '\\\
        int                m_nResFiles;
        BOOL                m_bMute; // Turn Sound Off
        int                m_nBubbleTextLimit; // Limit to the length
of the text in a bubble
        int                m_nBubbleTime; // in millisec
        // SATYA CHANGES START
        int                m_nChnCatTree;
        CChnCatTree*       m_pChnCatTree;
        CInternetSession *m_pInetSes;
        CFtpConnection     *m_pFtpCon;
        // SATYA CHANGES END
private:
        //CHANGES_MADE_FOR_UNICHAT_2
        int m_iMainCat; // Room main Catigory
        int m_iSubCat; // room sub catigorey
        //
        BOOL DownloadDynamicBannerInfo();
        CString m_strDynamicBannerLocalPath;

public:
        void SetRoomCatigories(int iMain,int iSub);
        BOOL CResMan::IsDynamicBanner(int nResId)
{

```


ResMan.h

```
        if (nResId == 312 || nResId == 313 || nResId == 315 || nResId == 20 ||  
nResId == 19)  
            return TRUE;  
        else  
            return FALSE;  
    }  
};  
  
#endif // __RESMAN_H
```

```

//      ResMan: Resource Manager
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT.COM
//=====
//
/* =====
; "resrc000.rit" for resrc000.bmc
;=====
; Resource Info Specification
; RIT: Resource Information Table
; Specification written by KEN Kim
; Feb 1998, UNICHAT
;=====
; filename=[(col,row)][,(cz.x,cz.y)][,nickname];
; filename = resource type flag(1) + name(4) + serial(3);
; resource type flag = t(TILE), s(STATIC), i(ANI), a(AVATAR), w(WAVE),
m(MIDI)
; (col,row) = # of cells in (column, row); for STATIC this attribute is
omitted
; (cz.x,cz.y) = center of z-order in (x, y) pixel position from the left-
top
; nickname = nickname for actor

// Think of tile as a PhasedSprite
#TIL= // TILE
{
    tcity001=(1,4);
    ...
}
#STT= // STATIC Sprites
{
    ccity002=(23,28); // Earth Position
    ...
}
#ANI= // ANIMATED Sprites
{
    wroom001=(1,17),(34,8); // 34,22
    iadv000=(1,3),(123,321); // not exist
    ...
}
#AVT= // AVATAR
{
    aman_001=(11,4),(27,45),°Å°İ;
    ...
}
#SND= // SOUND
{
    wwhip; // wwhip.wav
    mtest; // mtest.mid
}

// Comments
// Resources are sequentially numbered according to the order in RIT.
// BMP file merger will read this info and then load each bitmap file

```

```

// with the same name as it's id (+.bm).

; ACTOR SECTION
; (*i): - change orientation, * 50%, / 25%, | 12%, ~ 0%, ^ don't USE_COLORKEY
; #ACTORBASE=nMSPT      // Basic formula for all the actors
; #ACTOR=id             // Overridden formula for special cases
; {
;     // nMSPT is Milliseconds per Tick
;     // Behaviors
;     0=nRepeatCount, (cell index(, delay ticks(, displacement_x,
displacement_y(, sound id)))...;
;     // nRepeatCount = 0 for infinite loop, - value for pendulum movement
;     // delay ticks; 5 = fast; 10 = moderate; 15 = slow
; }
; #COACTOR=...

#ACTORBASE=
{
    STANDF          =1, (0);
    STANDB          =1, (8);
    ...
}

#ACTOR=aman_000;
{
    SPECIAL          =1, ...
    ...
}
=====*/

#include "stdafx.h"

#include "ResMan.h"
#include "Parser.h"

#include "UC2Ani/DIB.h"
#include "UC2Ani/PhSprite.h"
#include "Behavior.h"
#include "Actor.h"

extern CParser gParser;

const char* RESFILE_FILTER = "Resource Information Tables(*.RIT)|*.RIT|"
                             "All files (*.*)|*.*|";
const char* RES_BMPEXT = ".bmp";    // ".bm"

static const PALETTEENTRY apeMASTER[256] =
{
#include "UC2Master.pal"    // 256 Color Table
};

////////////////////////////////////

WORD CResMan::CSpriteInfo::GetType()
{
    switch (tolower(m_strName[0]))
    {

```

```

        case 't':    return SPRITE_TILE;
        case 'w':    return SPRITE_WALL;
        case 'c':    return SPRITE_STATIC;
        case 'i':    return SPRITE_PHASED;
        case 'a':    return SPRITE_ACTOR;
    }
    return -1;
}

////////////////////////////////////

CResMan::CResMan()
{
    m_nSprites      = 0;
    m_aSI           = NULL;
    m_nActorDescs   = 0;
    m_aActorDesc    = NULL;
    m_nResFiles     = 1;
    m_strResFile.Empty();
}

CResMan::~CResMan()
{
    DeleteResources();
}

void CResMan::DeleteResources()
{
    if (m_aSI)
        delete [] m_aSI;
    m_aSI = NULL;
    m_nSprites = 0;
    if (m_aActorDesc)
        delete [] m_aActorDesc;
    m_aActorDesc = NULL;
    m_nActorDescs = 0;
    RemoveAllDIBs();
}

void CResMan::CountItems(CStdioFile& f, char* szBuf)
{
    m_nSprites = 0;
    int nAD=0;
    while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
    {
        gParser.CopyBuffer(szBuf);
        if (gParser.IsCommentLine()) // At first, check if it's a
comment line
            continue;
        if (!gParser.SetLeftToken('#') ||
            !gParser.GetValueRightToken(szBuf, '='))
            continue; // get next line
        if ((lstrcmpi(szBuf, "TIL") == 0) || // matching!
            (lstrcmpi(szBuf, "STT") == 0) ||
            (lstrcmpi(szBuf, "ANI") == 0) ||
            (lstrcmpi(szBuf, "AVT") == 0))
        {

```

```

        m_nSprites += CountContents(f, szBuf);
    }
    else if (lstrcmpi(szBuf, "ACTOR") == 0)    // matching!
    {
        m_aActorDesc[nAD++].SetNumBehaviors(CountContents(f,
szBuf));
    }
    }
    f.SeekToBegin();
}

// Count the number of lines that matches a specified string between '#' and
// '='
int CResMan::CountHeaders(CStdioFile& f, char* szBuf, const char* szToken)
{
    int i=0;
    while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
    {
        gParser.CopyBuffer(szBuf);
        if (gParser.IsCommentLine())    // At first, check if it's a
comment line
            continue;
        if (!gParser.SetLeftToken('#') ||    // get next line
!gParser.GetValueRightToken(szBuf, '='))
            continue;
        if (lstrcmpi(szBuf, szToken) == 0)    // matching!
            i++;
    }
    f.SeekToBegin();
    return i;
}

// Count the number of lines that include '=' between { and }
int CResMan::CountContents(CStdioFile& f, char* szBuf)
{
    int i=0;
    while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
    {
        gParser.CopyBuffer(szBuf);
        if (gParser.IsCommentLine())    // At first, check if it's a
comment line
            continue;
        if (gParser.SetLeftToken('{')    // Begin
        {
            if (gParser.SetLeftToken('}'))    // {}
                break;
            continue;    // get next line
        }
        if (gParser.SetLeftToken('}'))    // End of contents
            break;    // out of while loop
        if (gParser.SetLeftToken('='))
            i++;    // Anyway, we should increase the counter
    }
    return i;
}

BOOL CResMan::Initialize(CStdioFile& f, char* szBuf)

```

```

{
    DeleteResources();

    m_nActorDescs = CountHeaders(f, szBuf, "ACTOR");
    //////////// Initialize Actor Behaviors
    if (m_nActorDescs <= 0)
    {
        AfxMessageBox("Resource file: No behaviors found!");
        return FALSE;
    }
    m_aActorDesc = new CActorDesc[m_nActorDescs];
    TRACE("# of Actor Descriptions = %d\n", m_nActorDescs);

    CountItems(f, szBuf);

    //////////// Initialize Sprite Informations
    if (m_nSprites <= 0)
    {
        AfxMessageBox("Resource file: No sprites found!");
        return FALSE;
    }
    m_aSI = new CSpriteInfo[m_nSprites];
    TRACE("# of Sprites = %d\n", m_nSprites);

    return TRUE;
}

BOOL CResMan::SetResPath(LPCSTR path)
{
    m_strResPath = path;
    int len = lstrlen(path);
    if (path[len-1] != '\\')
        m_strResPath += "\\";
    return TRUE;
}

// Modify this so that it can handle multiple RIT files (resrc00?.rit)
BOOL CResMan::Load(LPCSTR path)
{
    if ((path == NULL) || (lstrlen(path) == 0))
    {
        ::SetCurrentDirectory(m_strResPath);
        // Show an Open File dialog to get the name.
        CFileDialog dlg(TRUE, // Open
            NULL, // No default extension
            m_strResFile, // Initial file name
            OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
            RESFILE_FILTER);

        if (dlg.DoModal() == IDOK)
            m_strResFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        m_strResFile = path;
    }
}

```

```

        IncludePath(m_strResFile);
    }

    char* szBuf = new char[gParser.GetMaxBuffer()]; // ReadString(CString&)
    has a bug!

    TRY
    {
        CFile f;
        if (!f.Open(m_strResFile, CFile::modeRead |
CFile::shareDenyWrite))
        {
            TRACE1("File Open Failure: %s\n", m_strResFile);
            delete [] szBuf;
            return FALSE;
        }
        UINT hLZFile = ::LZInit(f.m_hFile);
        LONG lFileSize = ::LZSeek(f.m_hFile, 0L, 2);    // points to the
end of file
        TRACE("FileSize=%ld bytes\n", lFileSize);
        ::LZSeek(hLZFile, 0, 0); // seek to beginning of file
        char* pTextSrc = new char[lFileSize+1];
        int iBytes = ::LZRead(hLZFile, pTextSrc, lFileSize);
        if (iBytes != (int)lFileSize)
        {
            TRACE("File read failure! %s: %d bytes read\n",
m_strResFile, iBytes);
            delete [] szBuf;
            return FALSE;
        }
        ::LZClose(hLZFile);

        // CStdioFile f(m_strResFile, CFile::modeRead | CFile::shareDenyNone
| CFile::typeText);
        ExcludePath(m_strResFile);

        if (!Initialize(f, szBuf))
        {
            f.Close();
            delete [] szBuf;
            return FALSE;
        }

        int i=0;
        int ad=0;
        CString strTemp;

        while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
        {
            gParser.CopyBuffer(szBuf);
            if (gParser.IsCommentLine())
                continue;
            if (!gParser.SetLeftToken('#'))
                continue;
            if (!gParser.GetValueRightToken(szBuf, '='))
                continue;

```

```

////////////////////////////////////
ENVIRONMENT SECTION
if (lstrcmpi(szBuf, "TIL") == 0)    // matching!
{
    while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
    {
        gParser.CopyBuffer(szBuf);
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken(''))
            break;
        if (!gParser.GetValueRightToken(strTemp, '='))
            continue;
        m_aSI[i].m_strName = strTemp;
        gParser.GetValueRightToken(m_aSI[i].m_sColRow,
            ',');

        m_aSI[i].m_sEarth = CSize(64/2,0);
        i++;
    }
}
else if (lstrcmpi(szBuf, "STT") == 0)
{
    while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
    {
        gParser.CopyBuffer(szBuf);
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken(''))
            break;
        if (!gParser.GetValueRightToken(strTemp, '='))
            continue;
        strTemp.MakeLower();
        m_aSI[i].m_strName = strTemp;
        m_aSI[i].m_sColRow = CSize(1, 1);
        gParser.GetValueRightToken(m_aSI[i].m_sEarth,
            ';');

        i++;
    }
}
else if (lstrcmpi(szBuf, "ANI") == 0)
{
    while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
    {
        gParser.CopyBuffer(szBuf);
        if (gParser.IsCommentLine())

```



```

        continue;
    if (gParser.SetLeftToken('{'))
    {
        if (gParser.SetLeftToken('}'))
            break;
        continue;
    }
    if (gParser.SetLeftToken('}'))
        break;
    if (!gParser.GetValueRightToken(strTemp, '='))
        continue;
    strTemp.MakeLower();
    m_aSI[i].m_strName = strTemp;
    gParser.GetValueRightToken(m_aSI[i].m_sColRow,

    gParser.GetValueRightToken(m_aSI[i].m_sEarth,

    i++;
}
}
else if (lstrcmpi(szBuf, "AVT") == 0)
{
    while (f.ReadString(szBuf, gParser.GetMaxBuffer()))
    {
        gParser.CopyBuffer(szBuf);
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken('}'))
            break;
        if (!gParser.GetValueRightToken(strTemp, '='))
            continue;
        strTemp.MakeLower();
        m_aSI[i].m_strName = strTemp;
        gParser.GetValueRightToken(m_aSI[i].m_sColRow,

        gParser.GetValueRightToken(m_aSI[i].m_sEarth,

        i++;
    }
}
else if (lstrcmpi(szBuf, "ACTOR") == 0)
{
    m_aActorDesc[ad++].Load(f, szBuf);
    if (ad >= m_nActorDescs)
        break;
}
else
{
    TRACE1("Unknown Data type in Cast file(%s)!\n",
szBuf);
}

```

```

    }
    f.Close();
    delete [] szBuf;
#ifdef _DEBUG
    for (i=0; i < m_nSprites; i++)
        TRACE("%d\\t%s=(%d,%d), (%d,%d)\\n", i, m_aSI[i].m_strName,
m_aSI[i].m_sColRow.cx, m_aSI[i].m_sColRow.cy,
        m_aSI[i].m_sEarth.cx, m_aSI[i].m_sEarth.cy);
    for (i=0; i < ad; i++)
        TRACE("%d\\t%d behaviors\\n", i,
m_aActorDesc[i].GetNumBehaviors());
#endif
    return TRUE;
}
CATCH( CFileException, e )
{
    #ifdef _DEBUG
        afxDump << "File could not be opened " << e->m_cause <<
"\n";
    #endif
    delete [] szBuf;
    return FALSE;
}
END_CATCH
}

int CResMan::GetSpriteID(const CString& strName) const
{
    for (int i=0; i < m_nSprites; i++)
    {
        if (lstrcmpi(m_aSI[i].m_strName, strName) == 0)
            return i;
    }
    return -1; // Not Found!
}

WORD CResMan::GetSpriteTypeByName(const CString& strName) const
{
    CString strRes(strName);
    MakeResName(strRes);
    int i = GetSpriteID(strRes);
    return (i >= 0) ? GetSpriteType(i) : 0;
}

////////////////////////////////////
// DIB Management
CDIB* CResMan::LoadDIB(LPCSTR szResName)
{
    CString strFile(szResName);
    ExpandResName(strFile, RES_BMPEXT);
    strFile.MakeLower();
    CDIB* pDIB;
    POSITION pos = m_oldDIB.GetHeadPosition();
    while (pos)
    {
        pDIB = (CDIB*)m_oldDIB.GetNext(pos); // Increment position.
        if (pDIB)

```

```

        {
            ASSERT(pDIB->IsKindOf(RUNTIME_CLASS(CDIB)));
            CString* pstr = pDIB->GetName();
            if (*pstr == strFile)    // compare with only filename
(excluding path)
                return pDIB;
        }
    }
    // File not found in current resource bank.
    // So create a new one and add it to the bank.
    pDIB = new CDIB;
    if (!pDIB->Load(strFile))
    {
        TRACE("%s: Resource Load Failure!\n", strFile);
        delete pDIB;
        return NULL;
    }
    MapToMasterPalette(pDIB);
    m_oldDIB.AddTail(pDIB);
    return pDIB;
}

CDIB* CResMan::LoadDIB(const WORD wResid)
{
    CString strName;
    strName.Format("RES:%d", wResid);
    CDIB* pDIB;
    POSITION pos = m_oldDIB.GetHeadPosition();
    while (pos)
    {
        pDIB = (CDIB*)m_oldDIB.GetNext(pos); // Increment position.
        if (pDIB)
        {
            ASSERT(pDIB->IsKindOf(RUNTIME_CLASS(CDIB)));
            CString* pstr = pDIB->GetName();
            if (*pstr == strName)
                return pDIB;
        }
    }
    // File not found in current resource bank.
    // So create a new one and add it to the bank.
    pDIB = new CDIB;
    if (!pDIB->Load(wResid))
    {
        TRACE("%d: Resource Load Failure!\n", wResid);
        delete pDIB;
        return NULL;
    }
    m_oldDIB.AddTail(pDIB);
    return pDIB;
}

// Load the specified resource into CPhasedSprite
// and sets initial conditions such as # of cells.
// The caller is responsible for deleting this sprite.
CPhasedSprite* CResMan::LoadPhasedSprite(LPCSTR szResName, const BOOL
bDIBReuse)

```

```

{
    CPhasedSprite* pPS;
    if (bDIBReuse) // Reuse Mechanism
    {
        CDIB* pDIB = LoadDIB(szResName);
        if (!pDIB)
            return NULL;
        pPS = new CPhasedSprite;
        pPS->SetDIB(pDIB); // link DIB resource to this sprite
    }
    else // Do not Reuse
    {
        CString strFile(szResName);
        ExpandResName(strFile, RES_BMPEXT);
        pPS = new CPhasedSprite;
        if (!pPS->Load(strFile))
        {
            delete pPS;
            TRACE("%s: Resource Load Failure!\n", strFile);
            return NULL;
        }
        ASSERT(pPS->GetDIB());
        MapToMasterPalette(pPS->GetDIB());
#ifdef MAPEDITOR
        if (GetSpriteType(GetSpriteID(szResName)) == SPRITE_ACTOR)
        {
            // Show Outline
            static int nColorSet = 1;
            RotateActorColorSet(pPS->GetDIB(), nColorSet);
            nColorSet++;
            if (nColorSet > 4)
                nColorSet = 1;
        }
#endif
    }
    #endif
    int i = GetSpriteID(szResName);
    ASSERT(i >= 0);
    pPS->SetType(GetSpriteType(i));
    pPS->SetNumCells(GetSpriteColRow(i));
    pPS->SetEarth(GetSpriteEarth(i));
    return pPS;
}

CActor* CResMan::LoadActor(const int nID)
{
    CActorDesc* pAD = GetActorDesc(nID);
    if (pAD)
    {
        CString strMsg;
        strMsg.Format("Actor Description: %d not found!", nID);
        AfxMessageBox(strMsg);
        return NULL;
    }

    CPhasedSprite* pPS = LoadPhasedSprite(*pAD->GetResName(), FALSE);
    if (!pPS)
        return NULL;
    CActor* pA = new CActor;

```

```

        CPhasedSprite* pPSDest = (CPhasedSprite*)pA;
        *pPSDest = *pPS;
        delete pPS;
        pA->SetID(nID);
        return pA;
    }

CBehavior* CResMan::GetActorBehavior(const int ad, const int bi)
{
    CActorDesc* pAD = GetActorDesc(ad);
    ASSERT(pAD);
    CBehavior* pBeh = pAD->GetBehavior(bi);
    if (!pBeh) // If not found in this ActorDesc, use that of the standard
instead
    {
        pAD = GetActorDesc(0); // Standard ActorDesc
        ASSERT(pAD);
        return pAD->GetBehavior(bi);
    }
    return pBeh;
}

void CResMan::DeleteDIB(CDIB* pDIB)
{
    POSITION pos = m_oldDIB.Find(pDIB);
    if (pos) // Hunt for pDIB
    {
        m_oldDIB.RemoveAt(pos);
        delete pDIB;
    }
}

// Be sure not to do use DIBs allocated here elsewhere!!!
void CResMan::RemoveAllDIBs()
{
    // Walk down the list deleting objects as we go.
    // We need to do this here because the base class simply deletes the
pointers.
    POSITION pos = m_oldDIB.GetHeadPosition();
    CDIB* pDIB;
    while (pos)
    {
        pDIB = (CDIB*)m_oldDIB.GetNext(pos); // Increment position.
        if (pDIB)
        {
            ASSERT(pDIB->IsKindOf(RUNTIME_CLASS(CDIB)));
            delete pDIB;
        }
    }
    // Now call the base class to remove the pointers.
    m_oldDIB.RemoveAll();
}

////////// Naming Handling Methods
// Extract filename only
void CResMan::MakeResName(CString& strName) const
{

```

```

    int i = strName.ReverseFind('\\');
    if (i >= 0)
    {
        char* p = strName.GetBuffer(256);
        int j = strName.ReverseFind('.');
        if (j > 0)
            p[j] = NULL;          // Exclude file extension
        p += (i+1);
        CString strTemp((char*)p);
        strName.ReleaseBuffer();
        strName = strTemp;
    }
}

void CResMan::ExcludePath(CString& strName) const
{
    int i = strName.ReverseFind('\\');
    if (i >= 0)
    {
        char* p = strName.GetBuffer(256);
        p += (i+1);
        CString strTemp((char*)p);
        strName.ReleaseBuffer();
        strName = strTemp;
    }
}

void CResMan::IncludePath(CString& strName) const
{
    if (strName.Find('\\') >= 0)
        return;
    CString strFN(m_strResPath);
    strFN += strName;
    strName = strFN;
}

// tile0000:tcity000 ==> C:\UC2\Data\tile0000:tcity000.bmp
void CResMan::ExpandResName(CString& strName, LPCSTR szExt) const
{
    CString strFN(m_strResPath);

    if ((strName.Find('\\') >= 0) || (strName.Find('.') >= 0))
        MakeResName(strName);
    strFN += strName;
    if (szExt)
        strFN += szExt;
    strName = strFN;
}

// Palette Control
// Be sure to call these methods before calling MapColorsToPalette
UINT CResMan::MapToMasterPalette(CDIB* pDIB) const
{
    return pDIB->SetPaletteEntries(0, 256, (const
LPPALETTEENTRY)apeMASTER);
}

```

```
void CResMan::ShowOutline(CDIB* pDIB) const
{
    // Write Outline Color to Outline (Off) Index
    pDIB->SetPaletteEntries(237, 1, (const
LPPALETTEENTRY)&(apeMASTER[240]));
}

// nColorSet = 1,2,3,4,
void CResMan::RotateActorColorSet(CDIB* pDIB, const int nColorSet) const
{
    int a = nColorSet;
    int b = nColorSet;
    if (nColorSet > 4)
    {
        a = nColorSet / 4;
        b = nColorSet % 4;
    }
    pDIB->RotatePaletteIndex(200, 4*4, a*4);
    pDIB->RotatePaletteIndex(220, 4*4, b*4);
}
```

RESOURCE.H

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by UC2.rc
//
#define IDD_ABOUTBOX 100
#define CG_IDD_CONTROLPANEL 102
#define CG_ID_VIEW_CONTROLPANEL 103
#define IDP_SOCKETS_INIT_FAILED 104
#define IDS_CLOSE_TEXT 105
#define IDS_JOIN_CHANNEL_TITLE 106
#define IDS_TIME_REPORT 107
#define IDD_PP_MYINFO 107
#define IDS_ABOUT_TEXT 108
#define IDS_ERROR_TIMER 109
#define IDS_ERROR_BROWSER 110
#define IDS_UNICHAT_HOMEURL 111
#define CG_IDD_HISTORYPANEL 112
#define IDS_HISTORY_PANEL 113
#define CG_ID_VIEW_HISTORYPANEL 114
#define IDS_PROPSHT_CAPTION 115
#define IDD_PP_CREATE_CHANNEL 116
#define IDS_DATA_TRANSMISSION_DONE 116
#define IDD_PP_CHANNEL 117
#define IDS_TRANSFER_MEMBER_CHANNEL 117
#define IDD_PP_ACTOR 118
#define IDS_TRANSFER_MEMBER_ALL_CHANNEL 118
#define IDD_PP_MEMBER1 119
#define IDD_PP_MEMBER2 120
#define DIB_ROOMKEY 122
#define ID_INDICATOR_DATE 123
#define ID_INDICATOR_TIME 124
#define IDR_MAINFRAME 128
#define IDR_UC2TYPE 129
#define IDS_QUERY_LINE_BUSY 129
#define IDD_DIALOG_LOGIN 130
#define IDS_INVITE_SELECT_MEMBER 130
#define IDD_CLOSE 131
#define IDS_INVITE_OPEN_CHANNEL 131
#define IDR_MENU_ACTOR 132
#define IDS_INVITE_ERROR_MYSELF 132
#define IDB_IL_CHANNEL 133
#define IDC_ARROW_LT 134
#define IDS_INVITE_OK 134
#define IDC_ARROW_LB 135
#define IDS_INVITE_FAIL 135
#define IDC_ARROW_RT 136
#define IDS_JOINED_CHANNEL 136
#define IDC_ARROW_RB 137
#define IDS_KICKOFF_REASON 137
#define IDC_HGREP_DOWN 138
#define IDS_MEMBER_ENTRY 138
#define IDC_HGREP_UP 139
#define IDS_MEMBER_EXIT 139
#define IDD_DLG_MEMBER 140
#define IDS_INVITATION 140
#define IDS_PRIVATE_MESSAGE 141
#define IDD_INPUT_INT 141
```


RESOURCE.H

```

#define IDS_SELECT_CHANNEL 142
#define IDS_FILL_CHANNEL_TITLE 143
#define IDS_NO_CHANNEL_OPENED 144
#define IDI_UCICON 144
#define IDS_CLICK_CHANNEL_TITLE 145
#define IDC_CURSOR1 145
#define IDS_RECEIVING_CHANNELS_LIST 146
#define IDS_INVALID_CHANNEL_NAME 147
#define IDB_IL_CREATECHANNEL 147
#define IDS_LEAVE_THIS_CHANNEL 148
#define IDB_IL_MEMBER 149
#define IDS_CANNOT_JOIN_CHANNEL 149
#define IDS_INVALID_CHATID 150
#define IDS_CONNECTION_TYPE_GLOBAL 151
#define IDS_CONNECTION_TYPE_LOCAL 152
#define IDR_MENU_ACTOR_OTHER 167
#define IDD_INPUT_PASSWORD 169
#define IDD_DIALOG_WHISPER 172
#define IDD_PROGRESS_DIALOG 173
#define IDD_PP_MYADDRESS 175
#define IDD_PP_OTHERINFO 176
#define IDC_EDIT_MSG 1000
#define IDC_DISPLAY 1001
#define IDC_BTN_CREATE 1002
#define IDC_PROGRESS 1002
#define IDC_BTN_ROOM 1003
#define IDC_BTN_MEMBER 1004
#define IDC_BTN_SOUND 1005
#define IDC_BTN_HISTORY 1006
#define IDC_BTN_QUIT 1007
#define IDC_EDIT_NAME 1008
#define IDS_QUERY_NO_MATCH 1008
#define IDC_RADIO_LAN 1009
#define IDC_EDIT_CHAT_ID 1009
#define IDS_CONNECT_LAN 1009
#define IDC_RADIO_MODEM 1010
#define IDC_EDIT_UNITEL_ID 1010
#define IDS_CONNECT_MODEM 1010
#define IDC_EDIT_HOMEPAGE 1011
#define IDS_WELCOME_TO_LOBBY 1011
#define IDC_EDIT_NICKNAME 1012
#define IDC_EDIT_PASSWORD 1013
#define IDC_PROGRESSBAR 1013
#define IDC_EDIT_LNAME 1013
#define IDC_CB_HOST 1014
#define IDC_EDIT_PROFILE 1014
#define IDS_CANCELED_CONNECTION 1014
#define IDS_LOGIN_BEGIN 1015
#define IDC_LIST_CHANNEL 1015
#define IDC_EDIT_AGE 1015
#define IDS_LOGIN_CONNECTING 1016
#define IDC_SPIN_AGE 1016
#define IDC_EDIT_SEND 1017
#define IDC_ST_MESSAGE 1018
#define IDC_ST_TIME_REPORT 1019
#define IDC_LB_ACTORS 1019
#define IDS_CONNECTING_SERVER 1019

```

RESOURCE.H

```

#define IDC_EDIT_HISTORY 1020
#define IDC_SB_ACTOR 1020
#define IDS_LOGGING_ON_SERVER 1021
#define IDC_ST_SECONDS 1021
#define IDS_TRYING_BACKUP_ID 1022
#define IDC_ST_VERSION 1022
#define IDC_STATIC_NICK 1023
#define IDS_CONNECTION_FAILED 1023
#define IDC_BTN_MUD 1023
#define IDS_ERROR_TIMEOUT 1024
#define IDC_ST_STAGEID 1024
#define IDS_ERROR_NOT_WIN95 1025
#define IDC_LIST_MEMBER 1026
#define IDS_ERROR_WRITE_REGISTRY 1026
#define IDC_EDIT_INT 1026
#define IDC_STATIC_MESSAGE 1027
#define IDS_ENTER_PASSWORD 1027
#define IDC_BTN_DEFAULT 1027
#define IDC_BTN_RENEW 1028
#define IDS_DEMO_1 1028
#define IDC_EDIT_EMAIL 1028
#define IDC_STATIC_CHANNEL_COUNT 1029
#define IDS_DEMO_2 1029
#define IDC_STATIC_MEMBER_COUNT 1030
#define IDS_DEMO_3 1030
#define IDC_COMBO_SEX 1031
#define IDC_BTN_INVITE 1031
#define IDS_DEMO_4 1031
#define IDS_DEMO_5 1032
#define IDS_DEMO_6 1033
#define IDC_STATIC_REALNAME 1034
#define IDS_DEMO_7 1034
#define IDS_DEMO_8 1035
#define IDS_DEMO_9 1036
#define IDC_PUBLISH_EMAIL 1036
#define IDS_DEMO_10 1037
#define IDC_COMBO_ETHNIC 1037
#define IDC_ST_ALIAS 1037
#define IDS_DEMO_11 1038
#define IDC_COMBO_WORK 1038
#define IDS_DEMO_12 1039
#define IDC_ST_UNICHATID 1039
#define IDS_DEMO_13 1040
#define IDC_EDIT_FNAME 1040
#define IDC_EDIT_CHANNEL_NAME 1041
#define IDS_DEMO_14 1041
#define IDC_ST_EMAIL 1041
#define IDS_ACTOR_PROPSHT_CAPTION 1042
#define IDC_RADIO_PUBLIC 1043
#define IDS_SELECT_STAGE 1043
#define IDC_ST_FNAME 1043
#define IDC_RADIO_PRIVATE 1044
#define IDS_MUD_MODE 1045
#define IDS_EPILOGUE00 1046
#define IDC_CHANNEL_TREE 1046
#define IDS_EPILOGUE01 1047
#define IDC_ST_LNAME 1047

```

RESOURCE.H

```
#define IDS_EPILOGUE02 1048
#define IDC_CHANNEL_CAT_STATIC 1048
#define IDC_LB_BACKGROUND 1049
#define IDS_EPILOGUE03 1049
#define IDC_STATIC_LANGUAGE 1049
#define IDS_EPILOGUE04 1050
#define IDC_STATIC_VERSION 1051
#define IDS_PROGRESS_INIT_BIND 1051
#define IDC_ST_HOMEPAGE 1051
#define IDC_STATIC_UNITEL_ID 1052
#define IDS_PROGRESS_PREPARE 1052
#define IDC_STATIC_SEXAGE 1053
#define IDS_PROGRESS_DOWNLOAD_FAIL 1053
#define IDS_PROGRESS_DOWNLOADED 1054
#define IDC_ST_SEX 1054
#define IDC_EDIT_TEXT 1055
#define IDS_PROGRESS_ALL_FILES_DONE 1055
#define IDC_STATIC_TO 1056
#define IDS_CLOSE_ON_NEW_RIT 1056
#define IDC_CREATCHANNEL_TREE 1056
#define IDS_CHANNEL_OUT 1057
#define IDC_RAD_CRCNL_ENG 1057
#define IDS_RIT_NO_BEHAVIORS 1058
#define IDC_RAD_CRCNL_SPA 1058
#define IDS_RIT_NO_SPRITES 1059
#define IDC_ST_AGE 1059
#define IDC_RAD_CRCNL_KOR 1059
#define IDS_RIT_NO_WAVES 1060
#define IDC_RAD_CRCNL_CHI 1060
#define IDS_RIT_NO_MIDIS 1061
#define IDC_RAD_CRCNL_JAP 1061
#define IDS_RIT_NO_STAGES 1062
#define IDC_RAD_CRCNL_OTH 1062
#define IDS_RIT_NO_SERVERIPS 1063
#define IDC_ST_ETHNIC 1063
#define IDC_STATIC_PASSWORD 1063
#define IDS_ENTER_BUBBLE_TEXT_LIMIT 1064
#define IDS_ENTER_BUBBLE_TIME 1065
#define IDC_ST_WORK 1065
#define IDS_BTN_MUD 1066
#define IDC_BTN_USER 1067
#define IDS_HELP_SIMPLE 1068
#define IDC_ST_LANGUAGE 1070
#define IDC_EDIT_ADDR_STREET 1073
#define IDC_EDIT_TEL_MOBILE 1075
#define IDC_EDIT_TEL_FAX 1077
#define IDC_EDIT_TEL_HOME 1081
#define IDC_EDIT_ADDR_CITY 1107
#define IDC_CB_COUNTRY 1115
#define IDC_EDIT_TEL_WORK 1125
#define IDC_EDIT_TEL_OTHER 1129
#define IDC_EDIT_ADDR_ZIP 1143
#define IDC_EDIT_ADDR_STATE 1144
#define IDC_RADIO_ENG 1146
#define IDC_RADIO_KOR 1147
#define IDC_RADIO_CHI 1148
#define IDC_RADIO_JAP 1149
```

RESOURCE.H

```

#define IDC_RADIO_SPA 1150
#define IDC_RADIO_OTH 1151
#define ID_VIEW_ADJUST_WINDOW 32771
#define ID_VIEW_PAUSE 32772
#define ID_ACTOR_VOICE 32773
#define ID_ACTOR_MOVE_F 32774
#define ID_ACTOR_MOVE_B 32775
#define ID_ACTOR_TURN_L 32776
#define ID_ACTOR_TURN_R 32777
#define ID_ACTOR_STATE_0 32778
#define ID_ACTOR_STATE_1 32779
#define ID_ACTOR_STATE_2 32780
#define ID_ACTOR_PROP 32781
#define IDS_TURNING_OFF_DEMO 32782
#define ID_EDIT_NOTEPAD 32784
#define ID_HELP_HOMEPAGE 32785
#define ID_VIEW_DEMO 32786
#define ID_ACTOR_HYPERLINK 32787
#define ID_CONNECT_SYNC 32788
#define ID_VIEW_TEXT_LIMIT 32789
#define ID_VIEW_BUBBLE_TIME 32790
#define ID_VIEW_BUBBLE_TEXTLIMIT 32791
#define ID_ABOUT_DEVELOPERS 32792
#define ID_HELP_DEVELOPERS 32793
#define ID_ACTOR_ACTION_0 32805
#define ID_ACTOR_ACTION_1 32806
#define ID_ACTOR_ACTION_2 32807
#define ID_ACTOR_ACTION_3 32808
#define ID_ACTOR_ACTION_4 32809
#define ID_ACTOR_ACTION_5 32810
#define ID_ACTOR_ACTION_6 32811
#define ID_ACTOR_ACTION_7 32812
#define ID_ACTOR_ACTION_8 32813
#define ID_ACTOR_ACTION_9 32814
#define ID_MAKEHOST 32823
#define ID_WHISPER 32830
#define ID_IGNORE 32831
#define ID_KICKOUT 32832
#define ID_UCICON 32835
#define WM_UCICON_NOTIFY 32836
#define UCMN_MINIMISE 32837
#define UCMN_MAXIMISE 32838
#define UCMN_UNICHAT 32839
#define UCMN_EXIT 32840

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS 1
#define _APS_NEXT_RESOURCE_VALUE 148
#define _APS_NEXT_COMMAND_VALUE 32794
#define _APS_NEXT_CONTROL_VALUE 1070
#define _APS_NEXT_SYMED_VALUE 125
#endif
#endif

```

Splash.cpp

```
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT
//=====
// CG: This file was added by the Splash Screen component.
// Splash.cpp : implementation file
//

#include "stdafx.h" // e. g. stdafx.h
#include "resource.h" // e.g. resource.h

#include "Splash.h" // e.g. splash.h
#include "ResMan.h"

#include "UC2Ani/DIBPal.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

#ifdef _MALL
LPCTSTR BMP_SPLASH = "MSplash.bmp";
#else
LPCTSTR BMP_SPLASH = "U2Panel\\U2Splash.bmp";
#endif

////////////////////////////////////
//      Splash Screen class

BOOL CSplashWnd::c_bShowSplashWnd;
CSplashWnd* CSplashWnd::c_pSplashWnd;
CSplashWnd::CSplashWnd()
{
    TRACE0("CSplashWnd::CSplashWnd()\n");
    m_pPal = NULL;
    m_pDIB = NULL;
}

CSplashWnd::~CSplashWnd()
{
    TRACE0("CSplashWnd::~CSplashWnd()\n");
    if (m_pDIB)
        delete m_pDIB;
    if (m_pPal)
        delete m_pPal;
    // Clear the static window pointer.
    ASSERT(c_pSplashWnd == this);
    c_pSplashWnd = NULL;
}

BEGIN_MESSAGE_MAP(CSplashWnd, CWnd)
    //{{AFX_MSG_MAP(CSplashWnd)
```

```

        ON_WM_CREATE()
        ON_WM_PAINT()
        ON_WM_TIMER()
        ON_WM_QUERYNEWPALETTE()
        ON_WM_PALETTECHANGED()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CSplashWnd::EnableSplashScreen(BOOL bEnable /*= TRUE*/)
{
    c_bShowSplashWnd = bEnable;
}

void CSplashWnd::ShowSplashScreen(CWnd* pParentWnd /*= NULL*/)
{
    if (!c_bShowSplashWnd || c_pSplashWnd != NULL)
        return;

    // Allocate a new splash screen, and create the window.
    c_pSplashWnd = new CSplashWnd;
    if (!c_pSplashWnd->Create(pParentWnd))
    {
        delete c_pSplashWnd;
        c_pSplashWnd = NULL;
    }
    else
    {
        c_pSplashWnd->UpdateWindow();
    }
}

BOOL CSplashWnd::PreTranslateAppMessage(MSG* pMsg)
{
    if (c_pSplashWnd == NULL)
        return FALSE;

    // If we get a keyboard or mouse message, hide the splash screen.
    if (pMsg->message == WM_KEYDOWN ||
        pMsg->message == WM_SYSKEYDOWN ||
        pMsg->message == WM_LBUTTONDOWN ||
        pMsg->message == WM_RBUTTONDOWN ||
        pMsg->message == WM_MBUTTONDOWN ||
        pMsg->message == WM_NCLBUTTONDOWN ||
        pMsg->message == WM_NCRBUTTONDOWN ||
        pMsg->message == WM_NCMBUTTONDOWN)
    {
        c_pSplashWnd->HideSplashScreen();
        return TRUE;        // message handled here
    }

    return FALSE;        // message not handled
}

BOOL CSplashWnd::Create(CWnd* pParentWnd /*= NULL*/)
{
    TRACE("CSplashWnd::Create(%lx)\n", pParentWnd);
    CString strFile(*gResMan.GetResPath());

```

Splash.cpp

```
// CString strPal(strFile);
// strFile += BMP_SPLASH;
// strPal += "U2Login.pal";

// CFileStatus status;
// BOOL bFound = CFile::GetStatus(strFile, status);
// if (!bFound)
//     return FALSE;    // To avoid file not found message dialog box

m_pDIB = new CDIB;
if (!m_pDIB->Load(strFile)//, strPal))
{
    delete m_pDIB;
    m_pDIB = NULL;
    return FALSE;
}

#ifdef _MALL
    gResMan.LoadMasterPalette(m_pDIB);
#endif

// Create the palette from the DIB.
m_pPal = new CDIBPal;
ASSERT(m_pPal);
if (!m_pPal->Create(m_pDIB))
{
    AfxMessageBox("Failed to create palette from DIB file");
    delete m_pPal;
    m_pPal = NULL;
}

return CreateEx(0,
    AfxRegisterWndClass(0, AfxGetApp()-
>LoadStandardCursor(IDC_ARROW)),
    NULL, WS_POPUP | WS_VISIBLE, 0, 0, m_pDIB->GetWidth(), m_pDIB-
>GetHeight(),
    pParentWnd->GetSafeHwnd(), NULL);
}

void CSplashWnd::HideSplashScreen()
{
    TRACE0("CSplashWnd::HideSplashScreen()\n");
    // Destroy the window, and update the mainframe.
    if (c_pSplashWnd)
    {
        c_pSplashWnd->DestroyWindow();
        AfxGetMainWnd()->UpdateWindow();
    }
}

void CSplashWnd::PostNcDestroy()
{
    // Free the C++ class.
    delete this;
}

int CSplashWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)

```

```

        return -1;

        // Center the window.
        CenterWindow();

        // Set a timer to destroy the splash screen.
        // SetTimer(1, 750, NULL);

        return 0;
    }

void CSplashWnd::OnPaint()
{
    CPaintDC dc(this);

    // Make sure we have what we need to do a paint.
    if (!m_pDIB)
    {
        TRACE("No DIB or color table to paint from.\n");
        return;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    if (m_pPal)
    {
        pPalOld = dc.SelectPalette(m_pPal, FALSE);        //
        bForceBackground = FALSE
        // dc.RealizePalette();    // we realize in response to
        WM_QUERYNEWPALETTE
    }
    m_pDIB->Draw(&dc, 0, 0);
    // Select old palette if we altered it.
    if (pPalOld)
        dc.SelectPalette(pPalOld, FALSE);
    }

void CSplashWnd::OnTimer(UINT nIDEvent)
{
    // Destroy the splash screen window.
    HideSplashScreen();
}

void CSplashWnd::OnPaletteChanged(CWnd* pFocusWnd)
{
    // CDialog::OnPaletteChanged(pFocusWnd);
    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CSplashWnd::OnQueryNewPalette()
{
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE);    //
        foreground
    }
}

```


Splash.cpp

```
UINT u = pdc->RealizePalette();
if (pPalOld)
    pdc->SelectPalette(pPalOld, FALSE);
ReleaseDC(pdc);
if (u)
{
    // Some colors changed so we need to do a repaint.
    Invalidate(); // Repaint the lot.
    return TRUE; // Say we did something.
}
return FALSE; // Say we did nothing.
//
return CDialog::OnQueryNewPalette();
}
```

Splash.h

```
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT
//=====
// CG: This file was added by the Splash Screen component.

#ifndef _SPLASH_SCRN_
#define _SPLASH_SCRN_

// Splash.h : header file
//

////////////////////////////////////

// Splash Screen class
class CDIB;
class CDIBPal;

class CSplashWnd : public CWnd
{
// Construction
protected:
    CSplashWnd();

// Attributes:
public:

// Operations
public:
    static void EnableSplashScreen(BOOL bEnable = TRUE);
    static void ShowSplashScreen(CWnd* pParentWnd = NULL);
    static void HideSplashScreen();
    static BOOL PreTranslateAppMessage(MSG* pMsg);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSplashWnd)
    //}}AFX_VIRTUAL

// Implementation
public:
    ~CSplashWnd();
    virtual void PostNcDestroy();

protected:
    BOOL Create(CWnd* pParentWnd = NULL);
    static BOOL c_bShowSplashWnd;
    static CSplashWnd* c_pSplashWnd;

    CDIB*      m_pDIB;
    CDIBPal*   m_pPal;

// Generated message map functions
protected:
    //{{AFX_MSG(CSplashWnd)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnPaint();
    //}}AFX_MSG
};
```

Splash.h

```
afx_msg void OnTimer(UINT nIDEvent);
afx_msg BOOL OnQueryNewPalette();
afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};
```

```
#endif
```

Stage.cpp

```

// Stage.cpp: implementation of the CStage class.
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT
//=====

/* =====
; Stage stage000.sit
#VERSION=0.90;
#DEPENDENCY=
{
    resrc000.rit;
}

#STAGE=[u2] °Å°I °eÈ-¹æ Å\,ñ, master.pal, map000.mit;
{
//      resurce name=cell id, LeftTop(x,y), Image Operation, Elevation, Sprite
Type, nMSPT;
    ccity014=0, (333, -10), 4096, 8, http://www.microsoft.com;
    aman_007=26, (325, 67), 4096, 0, 272, 150;
; ...
} // 0 sprites.
===== */

#include "stdafx.h"
#include "Stage.h"

#include "TileMap.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/PhSprite.h"
#include "UC2Ani/Bubble.h"
#include "UC2Ani/OSBView.h"
#include "UC2Ani/MCIObj.h"
#include "Actor.h"
#include "ResMan.h"
#include "Parser.h"
#include "Behavior.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

extern CResMan    gResMan;                // global Resource Manager
extern CParser    gParser;

double GetMapEditorVersion();

const char* STAGEFILE_FILTER = "Stage Information Tables(*.SIT)|*.SIT|"
                                "All files (*.*)|*.*|";
const char* STAGEFILE_EXT = ".sit";        // should be lower case
////////////////////////////////////
// Construction/Destruction

```

```

////////////////////////////////////
#define DYNAMIC_BANNER 9999

CStage::CStage()
{
    TRACE0("CStage::CStage()\n");
    m_pOSBView      = NULL;
    m_pTiles        = NULL;
    m_pThisActor    = NULL;
    m_wRM           = RM_NORMAL;
    m_aBGM          = NULL;
    m_nBGMS         = m_iBGM = 0;
    m_pBGM          = NULL;
    m_strStageFile.Empty();
    m_strPalFile.Empty();
    m_bExitOpen     = FALSE;
    m_bMusicPlayOK  = FALSE;
}

CStage::~CStage()
{
    TRACE0("CStage::~CStage()\n");
    DeleteStage();
}

////////////////////////////////////
// Be sure to call this function before calling any other member functions
// pass COSBView-inherited View
BOOL CStage::Initialize(COSBView* pView)
{
    m_pOSBView = pView;
    m_SpriteList.m_NotifyObj.SetView(pView);
    // m_AnimeType.m_NotifyObj.SetView(pView); // Each sprite need not
    notify
    m_BubbleList.m_NotifyObj.SetView(pView);
    return TRUE;
}

void CStage::DeleteStage()
{
    TRACE0("CStage::DeleteStage()\n");
    m_pThisActor = NULL;

    m_SpriteList.RemoveAll(); // This should go before deleting Tiles
    m_BubbleList.RemoveAll();
    m_AnimeType.RemoveAll(FALSE); // do not delete

    if (m_pTiles)
        delete m_pTiles;
    m_pTiles = NULL;

    if (m_aBGM)
        delete [] m_aBGM;
    m_aBGM = NULL;
    if (m_pBGM)

```

Stage.cpp

```

        delete m_pBGM;
        m_pBGM = NULL;

        m_strTitle.Empty();
        m_strMusicSeq.Empty();
        m_bExitOpen = FALSE;
    }

    BOOL CStage::InitMap()
    {
        if (m_pTiles)
            delete m_pTiles;
        m_pTiles = new CTileMap;
        m_pTiles->SetPalette((CPalette*)m_pOSBView->GetOSBPalette());
        m_pTiles->SetSpriteList(&m_SpriteList);
        m_pTiles->SetAniList(&m_AniList);
        ASSERT(m_pOSBView);

        return TRUE;
    }

    BOOL CStage::CreateStage(CSize& szT, CSize& szScr)
    {
        TRACE("CStage::CreateStage()\n");
        // CreateStage() function will call gResMan.Load() every time
        // to update (maybe) changed items in RIT file.
        if (!gResMan.Load())
            return FALSE;

        CreateOSB(szScr);
        ASSERT(m_pOSBView);

        InitMap(); // New m_pTiles
        m_pTiles->SetTileSize(szT);
        m_pTiles->Create(szScr);
#ifdef MAPEDITOR
        m_pTiles->LoadGrid();
#endif
        return TRUE;
    }

    // Create OSB from the Map file (tiles)
    BOOL CStage::CreateOSB(const CSize sScreen)
    {
        // Delete any existing sprites
        m_SpriteList.RemoveAll(); // Since the palette will be changed

        // Create OSB
        CDIB* pDIB = new CDIB;
        // Load master palette - This will be an OSB palette
        CString strPal(m_strPalFile);
        gResMan.IncludePath(strPal);
        pDIB->Create(sScreen.cx, sScreen.cy); //, strPal);
        gResMan.LoadMasterPalette(pDIB);
        m_pOSBView->CreateOSB(pDIB); // virtual function
        delete pDIB;
        return TRUE;
    }

```

```

}

// Load SIT (Stage Information Table) file
// This method can be called multiple times in the App.'s Document class.
BOOL CStage::Load(const char* szStage)
{
    TRACE("CStage::Load(%s)\n", szStage);
    if (!m_strStageFile.IsEmpty())
    {
        // Save Previous Stage ID
        m_strPrevID = m_strStageFile;
        gResMan.MakeResName(m_strPrevID);
    }

    if ((szStage == NULL) || (lstrlen(szStage) == 0))
    {
        ::SetCurrentDirectory(*gResMan.GetResPath());
        // Show an Open File dialog to get the name.
        CFileDialog dlg(TRUE, // Open
                        NULL, // No default extension
                        m_strStageFile, // Initial file name
                        OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
                        STAGEFILE_FILTER);

        if (dlg.DoModal() == IDOK)
            m_strStageFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        m_strStageFile = szStage;
        gResMan.ExpandResName(m_strStageFile, STAGEFILE_EXT);
    }
}

#ifdef MAPEDITOR
// CreateStage() function will call gResMan.Load() every time
// to update (maybe) changed items in RIT file.
gResMan.Load(); // "resrc000.rit");
#endif

CTextFileBuffer tfb(gParser.GetMaxBuffer());
if (!tfb.Load(m_strStageFile))
{
    return FALSE;
}
gResMan.ExcludePath(m_strStageFile);

// int ny=0; // Current Row of the tiles
double fVersion;

while (tfb.ReadString())
{
    gParser.CopyBuffer(tfb.GetString());
    if (gParser.IsCommentLine())
        continue;
    if (!gParser.SetLeftToken('#'))
        continue;
}

```

```

CString strBuf;
if (!gParser.GetValueRightToken(strBuf, '='))
    continue;

////////////////////////////////////
if (lstrcmpi(strBuf, "VERSION") == 0)
{
    gParser.SetLeftToken('=');
    gParser.GetValueRightToken(fVersion, ',');
    if (fVersion <= 0.90)
    {
        CString strMsg;
        strMsg.Format("SIT: Old Version %.2f", fVersion);
        AfxMessageBox(strMsg);
    }
}
//////////////////////////////////// STAGE
else if (lstrcmpi(strBuf, "STAGE") == 0)
{
    DeleteStage();    // Delete current
    InitMap();

    gParser.SetLeftToken('=');
    gParser.GetValueRightToken(m_strTitle, ',', ';');

    if (fVersion <= 0.99)    // Separate MIT file
    {
        TRACE0("This SIT an obsolete version with separate
MIT file!\n");

        gParser.GetValueRightToken(m_strPalFile, ',', ';');
        CString strMap;
        gParser.GetValueRightToken(strMap, ',', ';');
        if (!m_pTiles->Load(strMap))
        {
            delete m_pTiles;
            m_pTiles = NULL;
            strMap += " not found!";
            AfxMessageBox(strMap);
            return FALSE;
        }
    }
    else // Version 1.0 or above
    {
        gParser.GetValueRightToken(m_strMusicSeq, ',', ';');
        ParseBGMS();    // will deal with empty string case
    }

    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}')
                break;
            continue;
        }
    }
}

```



```

    }
    if (gParser.SetLeftToken('{}'))
        break;

    CString strResName;

/*
    if (fVersion <= 0.90)    // Resource ID as a filename
    {
        gParser.GetValueRightToken(strResName, '=');
        gResMan.PrefixUDS00(strResName);
    }
    else    // New Version
    {

*/

        int nResID;
        gParser.GetValueRightToken(nResID, '=');

        //CString* pStr, BannerName;
        // code for the Dyanmic Banner ...
        // How we will find it is aDyanmic banner
        // Followoing code is commented for Testing

        if (gResMan.IsDynamicBanner(nResID))
        {
            CString BannerName;
            //CHANGES_MADE_FOR_UNICHAT_2

            if (gResMan.GetLatestDynamicBanner(BannerName))
            {
                strResName = BannerName;
            }
            else
            {
                CString *pStr =
gResMan.GetSpriteName(nResID);

                if (!pStr)
                {
                    strResName.Format("Invalid
ResID=%d in Stage file!", nResID);

                    AfxMessageBox(strResName);
                    continue;
                }
                strResName = *pStr;
            }
        }
        else
        {
            CString *pStr =
gResMan.GetSpriteName(nResID);

```

```

        if (!pStr)
        {
            strResName.Format("Invalid ResID=%d
in Stage file!", nResID);

            AfxMessageBox(strResName);
            continue;
        }
        strResName = *pStr;
    }
    // See comments above

    int         nCellID;
    CPoint       ptLT;
    WORD  wImOp = DEFAULT_IO;
    int     nElev = 0;
    WORD  wType = SPRITE_STATIC;
    int     nMSPT = 0;
    CString strHyperlink;

    gParser.GetValueRightToken(nCellID, ',');
    gParser.GetValueRightToken(ptLT, ',', ';');
    if (!gParser.GetValueRightToken(wImOp, ',', ';'))
        wImOp = DEFAULT_IO;
    else
    {
        if (!gParser.GetValueRightToken(nElev, ',',
';'))
            nElev = 0;
        else
        {
            if (!gParser.GetValueRightToken(wType,
',', ';'))
                wType = SPRITE_STATIC;
            else
            {
                if
(!gParser.GetValueRightToken(nMSPT, ',', ';'))
                    nMSPT = 0;
                else
                {
                    if
(!gParser.GetValueRightToken(strHyperlink, ',', ';'))
                        strHyperlink.Empty();
                }
            }
        }
    }
    // Resource Allocation via Resource Manager
    BOOL bReuse = (wType & SPRITE_ACTOR != SPRITE_ACTOR);
    CPhasedSprite* pPS =
gResMan.LoadPhasedSprite(strResName, bReuse, nResID);
    if (!pPS)
    {
        strResName += " not found!";
    }

```

```

        AfxMessageBox(strResName);
        continue;
    }
    pPS->SetCell(nCellID);
    if (nElev)
    {
        pPS->SetElevation(nElev);
        ptLT.y -= nElev;
    }
    pPS->SetLT(ptLT);
    pPS->SetType(wType);    // This should come before Z-
ordering for SPRITE_TILE!
#ifdef MAPEDITOR
    if (pPS->GetSrcType() == SPRITE_ACTOR)
        pPS->SetSrcType(SPRITE_PHASED);
#endif

    if (fVersion < 1.01)
    {
        wImOp &= ~NO_COLORKEY;
        if (nMSPT && !pPS->GetAniType())
            nMSPT = 0;
    }
    pPS->SetImOp(wImOp);
    pPS->SetZByEarth();
    if (nMSPT > 0)
        pPS->SetMSPT(nMSPT);
    //
    // if (!strHyperlink.IsEmpty())
    //     pPS->SetHyperlink(strHyperlink);

    InsertSprite(pPS);    // Insert into Sprite List
}

////////////////////// TILE: After version 1.0, MIT is integrated into SIT
else if (lstrcmpi(strBuf, "TILESIZE") == 0)
{
    ASSERT(fVersion >= 1.00);
    gParser.SetLeftToken('=');
    CSize sT;
    gParser.GetValueRightToken(sT, ';');
    m_pTiles->SetTileSize(sT);
}
else if (lstrcmpi(strBuf, "SCREENSIZE") == 0)    // matching!
{
    ASSERT(fVersion >= 1.00);
    gParser.SetLeftToken('=');
    CSize sScr;
    gParser.GetValueRightToken(sScr, ';');
    m_pTiles->Create(sScr);    // Create Tiles
}
else if (lstrcmpi(strBuf, "ROW") == 0)
{
    ASSERT(fVersion >= 1.00);
    gParser.SetLeftToken('(');
    int ny;
    gParser.GetValueRightToken(ny, ')');
    if (!m_pTiles->LoadRow(tfb, ny, fVersion))
        continue;
}

```

```

        }
        else
        {
            TRACE1("Unknown Data type in Stage file(%s)!\n", strBuf);
        }
    }
#ifdef MAPEDITOR
    if (fVersion >= 1.00)
        m_pTiles->LoadGrid();
#endif
    // Insert Animated Tiles to m_AniList

    return TRUE;
}

// syntax: 0>1>2>3>...n>;
void CStage::ParseBGMS()
{
    m_bMusicPlayOK = FALSE; // Prevent access to musice seq array
    if (!m_pBGM)
        m_pBGM = new CMCIObject; // Create once
    if (m_aBGM)
        delete [] m_aBGM;
    gParser.CopyBuffer(m_strMusicSeq);
    m_nBGMS = gParser.CountOccurrencesUpto('>', ';');
    if (m_nBGMS <= 0)
    {
        TRACE("No BGMS found in this stage\n");
        m_nBGMS = 3; // set default;
        m_aBGM = new int[m_nBGMS];
        for (int i=0; i < m_nBGMS; i++)
            m_aBGM[i] = i;
    }
    else
    {
        m_aBGM = new int[m_nBGMS];
        int nID;
        int i=0;
        while (gParser.GetValueRightToken(nID, '>', ';'))
        {
            ASSERT(i < m_nBGMS);
            m_aBGM[i++] = nID;
        }
    }
    m_iBGM = 0;
    m_bMusicPlayOK = TRUE;
}

// Paly BGMS sequentially
// Application's View window will drive this method
BOOL CStage::PlayBGM()
{
    if (!m_bMusicPlayOK)
        return FALSE;
    if (!m_pBGM)
    {
        TRACE("Error: BGM not created!\n");
    }
}

```

```

        return FALSE;
    }
    m_pBGM->Close();

    CString strFile(*gResMan.GetResPath());
    if (m_iBGM >= m_nBGMS)
        m_iBGM = 0;
    CString* pS = gResMan.GetMIDIName(m_aBGM[m_iBGM]);
    if (!pS)
    {
        TRACE("Resource(MIDI:%d) not found!\n", m_aBGM[m_iBGM]);
        return FALSE;
    }
    m_iBGM++;
    strFile += *pS;
    strFile += ".mid";

    if (!m_pBGM->Load(strFile))
        return FALSE;
    return m_pBGM->Play();
}

void CStage::StopBGM()
{
    if (m_pBGM)
        m_pBGM->Close();
}

// Save SIT (Stage Information Table) file
BOOL CStage::Save(const char* szStage)
{
    // Save Map File first
    if ((szStage == NULL) || (lstrlen(szStage) == 0))
    {
        ::SetCurrentDirectory(*gResMan.GetResPath());
        // Show a File Save dialog to get the name.
        CFileDialog dlg(FALSE, // Save
                        NULL, // No default extension
                        m_strStageFile, // No initial file
                        OFN_OVERWRITEPROMPT | OFN_HIDEREADONLY,
                        STAGEFILE_FILTER);
        if (dlg.DoModal() == IDOK)
            m_strStageFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        m_strStageFile = szStage;
        gResMan.ExpandResName(m_strStageFile, STAGEFILE_EXT);
    }

    TRY
    {
        ::CopyFile(m_strStageFile, "SIT.BAK", FALSE);
    }
}

```

```

        CStdioFile f(m_strStageFile, CFile::modeCreate | CFile::modeWrite
| CFile::typeText);
        gResMan.ExcludePath(m_strStageFile);

        CString str;
        str.Format("; Stage %s\n", m_strStageFile);
        f.WriteString(str);

        str.Format("#VERSION=%.2f;\n", GetMapEditorVersion());
        f.WriteString(str);

//        str.Format("#DEPENDENCY=\n{\n%s;\n}\n\n",
*gResMan.GetFileName());
//        f.WriteString(str);

        if (m_strTitle.IsEmpty())
            m_strTitle = "[u2]";
        str.Format("#STAGE=%s,%s;\n", m_strTitle, m_strMusicSeq);
        f.WriteString(str);
        f.WriteString("{\n");

        POSITION pos = m_SpriteList.GetTailPosition();
        CPhasedSprite* pPS;

        while (pos)
        {
            pPS = (CPhasedSprite*)m_SpriteList.GetPrev(pos);
            if (pPS->GetSrcType() == SPRITE_TILE)
                continue; // skip for tile
            m_SpriteList.RemoveRedundantSprite(pPS);
        }

        pos = m_SpriteList.GetTailPosition();
        int n=0;
        while (pos)
        {
            pPS = (CPhasedSprite*)m_SpriteList.GetPrev(pos);
            if (pPS->GetSrcType() == SPRITE_TILE)
                continue; // skip for tile

            CString strResName(*pPS->GetDIB()->GetName());
            gResMan.MakeResName(strResName);
            str.Format("%s=%d, (%d,%d), %d,%d,%d,%d;\n", strResName,
// #VERSION=0.90
            int nResID = gResMan.GetSpriteID(strResName);
            if (nResID < 0)
            {
                strResName += " not found";
                AfxMessageBox(strResName);
            }
            if ((pPS->GetImOp() == DEFAULT_IO) &&
                (pPS->GetElevation() == 0) &&
                (pPS->GetType() == SPRITE_STATIC))
            {
                (!pPS->HasHyperlink()))
            {
                // Typical case
                str.Format("%d=%d, (%d,%d)", nResID,
                    pPS->GetCellID(), pPS->GetX(),

```

```

        pPS->GetY() + pPS->GetElevation());
    }
    else
    {
        str.Format("%d=%d, (%d,%d),%d,%d,%d,%d", nResID,
            pPS->GetCellID(), pPS->GetX(),
            pPS->GetY() + pPS->GetElevation(),
            pPS->GetImOp(), pPS->
>GetElevation(),
            pPS->GetType(), pPS->GetMSPT());
//        if (pPS->HasHyperlink())
//        {
//            str += ',';
//            str += *pPS->GetHyperlink();
//        }
        str += ";\n";
        f.WriteString(str);
        n++;
    }
    str.Format("{}\t// %d sprites.\n", n);
    f.WriteString(str);

    str.Format("%s: %d sprites written.", m_strStageFile, n);
    if (m_pTiles)
    {
        if (!m_pTiles->GetFileName()->IsEmpty())
            str += "\n\nMIT file has been integrated into SIT
file.";

        m_pTiles->Save(f);
    }

    f.Close();
    AfxMessageBox(str);
}
CATCH( CFileException, e )
{
    #ifdef _DEBUG
        afxDump << "File could not be opened " << e->m_cause <<
"\n";
    #endif
    return FALSE;
}
END_CATCH
return TRUE;
}

////////////////////////////////////
// Sprite Manipulations

// Render the scene to the off-screen buffer pClipRect defaults to NULL
void CStage::Render(CRect* pClipRect)
{
    CRect rcDraw;
    ASSERT(m_posBView);
    CDIB* pOSB = m_posBView->GetOSB();
    ASSERT(pOSB);

```

```

    pOSB->GetRect(rcDraw);
    if (pClipRect)
        rcDraw.IntersectRect(pClipRect, &rcDraw);
//    if (m_pTiles)
//        m_pTiles->Render(pOSB, &rcDraw);    // Tiles are rendered with
sprites

    pOSB->ClearRect(rcDraw);    // No background image

    // Render the sprite list from the bottom of the list to the top
    POSITION pos = m_SpriteList.GetTailPosition();
    CSprite* pSprite;
    while (pos)
    {
        pSprite = m_SpriteList.GetPrev(pos);
        // while sprites off and for normal sprites, skip!
#ifdef MAPEDITOR
        if ((m_wRM & RM_NORMAL) == RM_NORMAL)
        {
#endif
            // MAPEDITOR
            pSprite->Render(pOSB, &rcDraw);
#ifdef MAPEDITOR
        }
        else
        {
            if ((m_wRM & RM_SPRITE) != RM_SPRITE)    // Exclude Sprite
            {
                if (pSprite->GetSrcType() != SPRITE_TILE)
                    continue;
            }
            else if ((m_wRM & RM_TILE) != RM_TILE)    // Exclude Tile
            {
                if (pSprite->GetSrcType() == SPRITE_TILE)
                    continue;
            }

            if ((m_wRM & RM_ELEV) != RM_ELEV)    // No Elevation
            {
                int nE = pSprite->GetElevation();
                if (nE)
                    pSprite->Unelevate();    // Temporarily do not
elevate

                pSprite->Render(pOSB, &rcDraw);
                if (nE)
                    pSprite->Elevate();
            }
        }
    }
#endif
    // MAPEDITOR
}

// Render the bubble list
pos = m_BubbleList.GetTailPosition();
CBubble* pBubble;
while (pos)
{
    pBubble = m_BubbleList.GetPrev(pos);
    // render only if this sprite hits the region
    pBubble->Render(m_pOSBView->GetHBitmap(),

```



```

(CPalette*)m_pOSBView->GetOSBPalette(),
&rcDraw);
}

#ifdef MAPEDITOR
    if (m_pTiles && ((m_wRM & RM_GRID) == RM_GRID))
        m_pTiles->RenderGrid(pOSB, &rcDraw);
#endif

#ifdef _DEBUG_RENDER
    static DWORD dwRender;
    dwRender++;
    if (dwRender%100 == 0)
        TRACE("CStage::Render %ld times...\n", dwRender);
#endif
    // _DEBUG
}
/*
void CStage::RenderZOrder(CSprite* pSprite)
{
    CDC dcMem;
    dcMem.CreateCompatibleDC(NULL);    // create a memory dc that is
compatible with current screen
    CBitmap* pbmOld = dcMem.SelectObject(CBitmap::FromHandle(m_pOSBView-
>GetHBitmap()));

    CPen pen(PS_SOLID, 1, PALETTE_RGB(255,0,0));
    CPen* penOld = (CPen*)dcMem.SelectObject(&pen);

    CPoint ptCtr(pSprite->GetCenter());
    CSize sScr(GetScreenSize());
    ptCtr.y = sScr.cy - pSprite->GetZ();

    const int d = 3;    // Draw a cross
    CRect rcZO(ptCtr.x-d, ptCtr.y-d, ptCtr.x+d,    ptCtr.y+d);
    dcMem.MoveTo(rcZO.TopLeft());
    dcMem.LineTo(rcZO.BottomRight());
    dcMem.MoveTo(rcZO.right, rcZO.top);
    dcMem.LineTo(rcZO.left, rcZO.bottom);

    m_pOSBView->AddDirtyRegion(&rcZO);

    dcMem.SelectObject(penOld);
    dcMem.SelectObject(pbmOld);
}
*/

// Multiple calls possible
void CStage::InsertSprite(CPhasedSprite* pPS)
{
    // At first remove if exist...
    m_AniList.Remove(pPS);
    m_SpriteList.Remove(pPS);
    m_SpriteList.Insert(pPS);
    if (pPS->GetAniType())
        m_AniList.Insert(pPS);    // add to the check list in
CStage::TickAll

```

```

        // map the colors in the sprite DIB to the
        // palette in the off-screen buffered view
        if (m_posBView->GetOSBPalette())
            pPS->MapColorsToPalette((CPalette*)m_posBView->GetOSBPalette());

        m_posBView->AddDirtyRegion(pPS);
    }

void CStage::DeleteSprite(CPhasedSprite* pPS)
{
    m_AniList.Remove(pPS);
    CPhasedSprite* pS = (CPhasedSprite*)m_SpriteList.Remove(pPS);
    if (pS)
    {
        m_posBView->AddDirtyRegion(pS);
        delete pS;
    }
}

// Multiple calls possible
void CStage::InsertBubble(CBubble* pBB)
{
    // At first remove if exist...
    m_BubbleList.Remove(pBB);
    m_BubbleList.Insert(pBB);
    m_posBView->AddDirtyRegion(pBB);
}

CBubble* CStage::RemoveBubble(CBubble* pBB)
{
    if (m_BubbleList.Remove(pBB))
    {
        m_posBView->AddDirtyRegion(pBB);
        return pBB;
    }
    return NULL;
}

// Insert Sprite into the animation sprites list
void CStage::InsertAniSprite(CPhasedSprite* pPS)
{
    // At first remove if exist...
    m_AniList.Remove(pPS);
    if (pPS->GetAniType())
        m_AniList.Insert(pPS); // add to the check list in
CStage::TickAll
}

// Create CActor and insert into the sprite list
CActor* CStage::CreateActor(const int nCharID, const CPoint& point,
                           const BOOL bTileID, const WORD
wState, const BOOL bThis)
{
    int nGCS=0; // Greatest ColorSet in this list
    for (POSITION pos = m_AniList.GetHeadPosition(); pos; )
    {
        CPhasedSprite* pPS = (CPhasedSprite*)m_AniList.GetNext(pos);
    }
}

```

```

        if (pPS->GetSrcType() == SPRITE_ACTOR)
        {
            CActor* pA = (CActor*)pPS;
            if (pA->GetCharID() == nCharID)
            {
                nGCS = max(nGCS, pA->GetColorSet()) + 1;
            }
        }
    }
    CActorDesc* pAD = gResMan.GetActorDesc(nCharID);
    if (!pAD)
    {
        CString strMsg;
        strMsg.Format("%d Actor is not defined in RIT.", nCharID);
        AfxMessageBox(strMsg);
        return NULL;
    }
    CActor* pActor = gResMan.LoadActor(nCharID, nGCS, bThis);
    if (!pActor)
        return NULL;
    CPoint ptTID;
    if (bThis) // point is ignored
    {
        m_pThisActor = pActor;
        ptTID = m_pTiles->GetEntryID(m_strPrevID); // Find Entry
    }
    else
    {
        ptTID = bTileID ? point : m_pTiles->GetNearestTileIndex(point);
    }
    if (!m_pTiles->IsValidTileID(ptTID))
    {
        TRACE("CreateActor - Invalid Tile ID [%d,%d]\n", ptTID.x,
ptTID.y);
        ptTID = CPoint(5,12);
    }
    CPoint ptC(m_pTiles->GetCenter(ptTID));
    int nEA = m_pTiles->GetEA(ptTID); // Elevation for actor
    if (nEA > ELEVATION_LIMIT)
    {
        TRACE("Actor was created in Elevation Limit Area!\n");
        pActor->SetOpacity(OPACITY50);
        nEA = m_pTiles->GetElevation(ptTID);
    }
    pActor->SetTileMap(m_pTiles);
    pActor->SetElevation(nEA);
    pActor->MoveToEarth(ptC);
    pActor->SetZByEarth();
    pActor->SetMSPT(pAD->GetMSPT());

    pActor->SetOpacity(OPACITY_0); // At first, hide this actor
    InsertSprite(pActor); // This is the first chance to draw sprite
    InsertBubble(pActor->GetBubble());

    // OPACITY_0 will be automatically cleared by the behavior
    descriptions.

```

Stage.cpp

```

    pActor->SetState(wState);      // First behavior

    TRACE("Actor(%lx)\n", pActor);
    return pActor;
}

CActor*      CStage::CreateActor(CMemberInfo& mi, const BOOL bThis)
{
    CActor* pA = CreateActor(mi.GetCharID(), mi.GetTileID(), TRUE,
                               mi.GetState(), bThis);

    ASSERT(pA);
    mi.SetTileID(pA->m_mi.GetTileID()); // CreateActor may have changed
tile ID
    pA->InitState(mi);
    return pA;
}

void CStage::DeleteActor(CActor* pActor)
{
    if (pActor == m_pThisActor)
        m_pThisActor = NULL;
    delete RemoveBubble(pActor->GetBubble());
    DeleteSprite((CPhasedSprite*)pActor);
}

/*
int CStage::SetZByYBottom(CPhasedSprite* pPS)
{
    //    CSize szScr(GetScreenSize());
    CPoint pt(pPS->GetEarthPoint());
    if (!m_pTiles)
        return (-pt.y);
    CPoint ptID(m_pTiles->GetNearestTileIndex(pt));
    return pPS->SetZByGroup(ptID.y);
}
*/
////////////////////////////////////
//////////
// PUMP, HEART!
// CPXClientApp::OnIdle calls this function during idle states between
message processing.
// TickAll activates all the actors and coactors in the stage.
int CStage::TickAll()
{
    DWORD dwCurTick = ::GetTickCount();
    int iCalled=0;
    POSITION pos = m_AniList.GetTailPosition();
    while (pos)
    {
        CPhasedSprite* pPS = (CPhasedSprite*)m_AniList.GetPrev(pos);
        // Increment position.
        ASSERT(pPS);
        if (dwCurTick >= pPS->GetAlarmTick())
        {
            if (pPS->HeartBeat(dwCurTick))
                m_posBView->RenderAndDrawDirtyList();
            iCalled++;
        }
    }
}

```

```

    }
}
pos = m_BubbleList.GetTailPosition();
while (pos)
{
    CBubble* pBB = m_BubbleList.GetPrev(pos);
    ASSERT(pBB);
    if (dwCurTick >= pBB->GetAlarmTick())
    {
        if (pBB->IsShown()) // Check if there is a bubble
            pBB->Show(FALSE); // Erase bubble
        iCalled++;
    }
}
// if (iCalled)
//     m_posBView->RenderAndDrawDirtyList();
return iCalled; // counter which indicates how are we busy...
}

// returns StageID on Exit tile
CString* CStage::ActorMove(CActor* pActor, const BOOL bForward)
{
    ASSERT(pActor);
    WORD wDA = pActor->GetDA();
    if (!bForward) // Backward
    {
        // Reverse direction
        if (wDA <= DA_FL) wDA <= 2;
        else wDA >= 2;
    }
    // CPoint ptTID(m_pTiles->GetNearestTileIndex(pActor->GetEarthPoint()));
    CPoint ptTID(pActor->m_mi.GetTileID());
    if (m_pTiles->GetActorNextTileID(ptTID, wDA)) // ptTID will be reset
    {
        CPoint ptC(m_pTiles->GetCenter(ptTID));
        int nEA = m_pTiles->GetEA(ptTID);
        pActor->MoveTo(ptC, nEA, bForward);

        CPhasedSprite* pTilePS = m_pTiles->GetPS(ptTID);
        if (m_bExitOpen &&
            pTilePS && (pTilePS->GetLinkType() ==
                CPhasedSprite::HLINK_U2_EXIT))
        {
            char szName[256];
            lstrcpy(szName, *pTilePS->GetHyperlink());
            ASSERT((szName[0] == 'x') && (szName[1] == ':'));
            m_strExitID = &szName[2];
            TRACE("Actor landed on Exit tile (%s)!\n", m_strExitID);
            return &m_strExitID;
        }
    }
    else
    {
        pActor->Act(CMD_SCRATCH);
        TRACE("Can't go further...\n");
    }
    return NULL;
}

```

```

void CStage::SetExitOpen(const BOOL bOpen)
{
    m_bExitOpen = bOpen;
#ifdef _MALL
    // Insert key sprites on the exit tiles
    CDIB* pDIBKey = gResMan.LoadDIB(DIB_ROOMKEY, TRUE);
    if (!pDIBKey)
    {
        TRACE0("Error: Could not create room key DIB!\n");
        return;
    }

    POSITION pos = m_SpriteList.GetTailPosition();
    CPhasedSprite* pPS;
    int n=0;
    while (pos)
    {
        pPS = (CPhasedSprite*)m_SpriteList.GetPrev(pos);
        if ((pPS->GetSrcType() == SPRITE_TILE) &&
            (pPS->GetLinkType() == CPhasedSprite::HLINK_U2_EXIT))
        {
            CPoint ptC(pPS->GetCenter());
            CPhasedSprite* pPSKey = new CPhasedSprite;
            pPSKey->SetDIB(pDIBKey);
            pPSKey->SetType(SPRITE_STATIC | SPRITE_ANI_FADE);
            switch (n++)
            {
            case 1:
                pPSKey->SetImOp(IMAGE_FLIP);
                pPSKey->SetMSPT(900);
                break;
            case 2:
                pPSKey->SetImOp(IMAGE_VERTICAL);
                pPSKey->SetMSPT(700);
                break;
            case 3:
                pPSKey->SetImOp(IMAGE_FLIP | IMAGE_VERTICAL);
                pPSKey->SetMSPT(500);
                break;
            default:
                pPSKey->SetMSPT(1000);
                break;
            }
            pPSKey->SetNumCells(1, 1);
            pPSKey->SetEarth(CPoint(10, 10));
            pPSKey->MoveToEarth(ptC);
            pPSKey->SetZByEarth();
            // Set Elevation
            InsertSprite(pPSKey);
        }
    }
#endif
}

// ChatSock-related
CActor* CStage::GetActor(PICS_MEMBER pMem)

```

```

{
    for (POSITION pos = m_AniList.GetHeadPosition(); pos; )
    {
        CPhasedSprite* pPS = (CPhasedSprite*)m_AniList.GetNext(pos);
        // Increment position.
        if (pPS->GetSrcType() == SPRITE_ACTOR)    // pPS-
        >IsKindOf(RUNTIME_CLASS(CActor))
        {
            CActor* pA = (CActor*)pPS;
            if (pA->m_mi.RefMember() == pMem)
                return pA;
        }
    }
    return NULL;
}

CActor* CStage::GetActor(CString& strNick)
{
    for (POSITION pos = m_AniList.GetHeadPosition(); pos; )
    {
        CPhasedSprite* pPS = (CPhasedSprite*)m_AniList.GetNext(pos);
        // Increment position.
        if (pPS->GetSrcType() == SPRITE_ACTOR)    // pPS-
        >IsKindOf(RUNTIME_CLASS(CActor))
        {
            CActor* pA = (CActor*)pPS;
            if (*pA->GetNick() == strNick)
                return pA;
        }
    }
    return NULL;
}

CActor* CStage::GetFrontActor()
{
    if (!m_pThisActor)
        return NULL;

    WORD wDA = m_pThisActor->GetDA();
    CPoint ptTID(m_pThisActor->m_mi.GetTileID());
    if (!m_pTiles->GetActorNextTileID(ptTID, wDA))    // ptTID will be reset
        return NULL;

    for (POSITION pos = m_AniList.GetHeadPosition(); pos; )
    {
        CPhasedSprite* pPS = (CPhasedSprite*)m_AniList.GetNext(pos);
        // Increment position.
        if (pPS->GetSrcType() == SPRITE_ACTOR)    // pPS-
        >IsKindOf(RUNTIME_CLASS(CActor))
        {
            if (((CActor*)pPS)->m_mi.GetTileID() == ptTID)
            {
                return (CActor*)pPS;
            }
        }
    }
    return NULL;
}

```

```

}

#ifdef MAPEDITOR
// Tile Manipulations
// Sprites in the map are not managed by CSpriteList class
// CSpriteList::Insert() method calls CSprite::SetNotificationObject().
// So the sprites handled by CSpriteList are automatically
// updates their dirty regions that are rendered in
// COSBView::RenderAndDrawDirtyList().
// But for the sprites in the map, we should handle dirty regions.

void CStage::InsertTile(CPhasedSprite* pPS)
{
    if (!m_pTiles)
        return;
    m_pTiles->Insert(pPS);
    if (m_posBView->GetOSBPalette())
        pPS->MapColorsToPalette((CPalette*)m_posBView->GetOSBPalette());

    m_posBView->AddDirtyRegion(pPS);
}

void CStage::DeleteTile(CPhasedSprite* pPS)
{
    if (!m_pTiles)
        return;
    m_posBView->AddDirtyRegion(pPS);

    m_pTiles->Delete(pPS);
}

BOOL CStage::MoveTileSpriteTo(CPhasedSprite* pPS, CPoint& ptTo)
{
    if (!m_pTiles)
        return FALSE;

    m_posBView->AddDirtyRegion(pPS);    // Save Current Region

    if (!m_pTiles->MoveSpriteTo(pPS, ptTo))
        return FALSE;

    m_posBView->AddDirtyRegion(pPS);
    return TRUE;
}
#endif

```


Stage.h

```
// Stage.h: interface for the CStage class.
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT.COM
//=====

#if !defined(AFX_STAGE_H_D5010EC4_A1F9_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_STAGE_H_D5010EC4_A1F9_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "UC2Ani/SpritLst.h"
#include "UC2Ani/BubblLst.h"
#include "TileMap.h"
#include "Actor.h"

////////////////////////////////////

class COSBView;
class CSprite;
class CPhasedSprite;
class CMCIObject;
class CMemberInfo;

enum RENDER_MODE
{
    RM_ELEV          = 0x0001,    // Show Elevation
    RM_TILE          = 0x0002,    // Show Tile
    RM_SPRITE        = 0x0004,    // Show Sprite
    RM_GRID          = 0x0008,
    RM_NORMAL        = RM_ELEV | RM_TILE | RM_SPRITE
};

class CStage : public CObject
{
public:
    CStage();
    virtual ~CStage();

// Attributes
    CSpriteList*      GetSpriteList()          { return &m_SpriteList; }
    int               GetNumSprites() const    { return
m_SpriteList.GetCount(); }
    int               GetNumAniSprites() const { return
m_AniList.GetCount(); }
    CTileMap*         GetTileMap()             { return m_pTiles; }
    CString*          GetFileName()            { return &m_strStageFile; }
    CString*          GetMapFileName()
                        { return (m_pTiles) ? m_pTiles->GetFileName() :
NULL; }
    CString*          GetPalFileName() { return &m_strPalFile; }
    CString*          GetTitle()              { return &m_strTitle; }
    CString*          GetMusicSeq()           { return &m_strMusicSeq; }
```

Stage.h

```

        CSize          GetScreenSize() const
                        { ASSERT(m_pTiles);      return m_pTiles-
>GetScreenSize(); }
        WORD          GetRenderMode() const { return m_wRM; }
        CActor*       GetThisActor()        { return m_pThisActor;
}
        CActor*       GetFrontActor();
        BOOL          IsExitOpen() const    { return m_bExitOpen; }

        CActor*       GetActor(CString& strNick);
        // ChatSock related
        CActor*       GetActor(PICS_MEMBER pMem);

// Operations
        BOOL          Initialize(COSBView* pView);
        void          DeleteStage();
        BOOL          CreateStage(CSize& szT, CSize& szScr);
        BOOL          CreateOSB(const CSize sScreen);
        BOOL          InitMap();
        BOOL          Load(const char* szStage=NULL);
        BOOL          Save(const char* szStage=NULL);
        void          Render(CRect* pClipRect=NULL);
        void          RenderZOrder(CSprite* pSprite);

        CSprite*      SpriteHitTest(const CPoint& point)
                        { return m_SpriteList.HitTest(point); }
        CPhasedSprite* TileHitTest(const CPoint& point)
                        { return (m_pTiles ? m_pTiles->HitTest(point) :
NULL); }
        CPhasedSprite* AnimatedHitTest(const CPoint& point)
                        { return
(CPhasedSprite*)m_AniList.HitTest(point); }

        void          InsertSprite(CPhasedSprite* pPS);
        void          InsertBubble(CBubble* pBB);
        void          InsertAniSprite(CPhasedSprite* pPS);
        void          DeleteSprite(CPhasedSprite* pPS);
        CBubble*      RemoveBubble(CBubble* pBB);
        void          RemoveAniSprite(CPhasedSprite* pPS)
                        { m_AniList.Remove((CSprite*)pPS); }

#ifdef MAPEDITOR
        void          InsertTile(CPhasedSprite* pPS);
        void          DeleteTile(CPhasedSprite* pPS);
        BOOL          MoveTileSpriteTo(CPhasedSprite* pPS, CPoint& ptTo);
#endif

        CActor*       CreateActor(const int nCharID, const CPoint&
point, const BOOL bTileID=TRUE,
                                const WORD wState=(AS_STAND |
DA_FR), const BOOL bThis=FALSE);
        CActor*       CreateActor(CMemberInfo& mi, const BOOL
bThis=FALSE);
        void          DeleteActor(CActor* pActor);

//      int          SetZByYBottom(CPhasedSprite* pPS);

        void          SetRenderMode(const WORD wRM) { m_wRM = wRM; }

```

Stage.h

```

        void                SetPalFileName(const CString& strPalFile) {
m_strPalFile = strPalFile; }
        void                ClearFilename()    { m_strStageFile.Empty(); }
        void                SetTitle(const CString& strTitle)    { m_strTitle =
strTitle; }
        void                SetMusicSeq(const CString& strMS)    { m_strMusicSeq =
strMS; }

        int                TickAll();
        BOOL                PlayBGM();
        void                StopBGM();

        CString*            ActorMove(CActor* pActor, const BOOL bForward);
        void                SetExitOpen(const BOOL bOpen=TRUE);

protected:
        void                ParseBGMS();

        COSBView*           m_pOSBView;
        CSpriteList          m_SpriteList;    // sprite list
        CSpriteList          m_AniList;        // Animated (Phased) Sprites,
Actors
        CBubbleList          m_BubbleList;
        CTileMap*            m_pTiles;
        CActor*              m_pThisActor;
        CString              m_strStageFile;    // filename.ext only
        CString              m_strPalFile;
        CString              m_strTitle;
        CString              m_strMusicSeq;
        int*                 m_aBGM;
        int                  m_nBGMS;    // Background Music
        int                  m_iBGM;      // Current BGM sequence #
0,1,2,...
        CMCIObject*          m_pBGM;        // MIDI sound
        WORD                 m_wRM;        // Render Mode
        BOOL                 m_bExitOpen;
        BOOL                 m_bMusicPlayOK;
        CString              m_strExitID;
        CString              m_strPrevID;
};

#endif //
!defined(AFX_STAGE_H__D5010EC4_A1F9_11D1_80E2_080009B9F339__INCLUDED_)

```

StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//      UC2.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information
```

```
#include "stdafx.h"
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H__A131386B_A610_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_STDAFX_H__A131386B_A610_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>             // MFC core and standard components
#include <afxext.h>             // MFC extensions
#include <afxdisp.h>           // MFC OLE automation classes
#ifdef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxsock.h>           // MFC socket extensions
#include <afxinet.h>

#include "UC2Messages.h"       // User-Defined Messages

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#endif(AFX_STDAFX_H__A131386B_A610_11D1_80E2_080009B9F339__INCLUDED_)
```

Test.cpp

```
// Test.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Test.h"

#include "MainFrm.h"
#include "TestDoc.h"
#include "TestView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CTestApp

BEGIN_MESSAGE_MAP(CTestApp, CWinApp)
    //{{AFX_MSG_MAP(CTestApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // NOTE - the ClassWizard will add and remove mapping macros
    here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

////////////////////////////////////
// CTestApp construction

CTestApp::CTestApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

////////////////////////////////////
// The one and only CTestApp object

CTestApp theApp;

////////////////////////////////////
// CTestApp initialization

BOOL CTestApp::InitInstance()
{
    if (!AfxSocketInit())
    {
        AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
        return FALSE;
    }
}
```

```

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC
statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings();    // Load standard INI file options (including
MRU)

    // Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CTestDoc),
        RUNTIME_CLASS(CMainFrame),           // main SDI frame window
        RUNTIME_CLASS(CTestView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

```

```

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
    // No message handlers
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CTestApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////
// CTestApp commands

```


Test.h

```
// Test.h : main header file for the TEST application
//

#if !defined(AFX_TEST_H__0BCFBAA6_B807_11D1_9169_444553540001__INCLUDED_)
#define AFX_TEST_H__0BCFBAA6_B807_11D1_9169_444553540001__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifdef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CTestApp:
// See Test.cpp for the implementation of this class
//

class CTestApp : public CWinApp
{
public:
    CTestApp();

    // Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTestApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

    // Implementation

    //{{AFX_MSG(CTestApp)
    afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //      DO NOT EDIT what you see in these blocks of generated code
    !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#endif(AFX_TEST_H__0BCFBAA6_B807_11D1_9169_444553540001__INCLUDED_)
```

A-459



Test.txt

tland000(0),tland000(1),tland000(5),tland000(0),tland000(1),
tland000(5),tland000(0),tland000(1),tland000(5),tland000(5)
tland000(0),tland000(1),tland000(5),tland000(0),tland000(1),
tland000(5),tland000(0),tland000(1),tland000(5),tland000(5)
tland000(0),tland000(1),tland000(5),tland000(0),tland000(1),
tland000(5),tland000(0),tland000(1),tland000(5),tland000(5)
tland000(0),tland000(1),tland000(5),tland000(0),tland000(1),
tland000(5),tland000(0),tland000(1),tland000(5),tland000(5)
tland000(0),tland000(1),tland000(5),tland000(0),tland000(1),
tland000(5),tland000(0),tland000(1),tland000(5),tland000(5)

TestDoc.cpp

```
// TestDoc.cpp : implementation of the CTestDoc class
//

#include "stdafx.h"
#include "Test.h"

#include "TestDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

const LPCTSTR UNICHAT_FTP="ftp://203.241.132.83/UniChat/";
/////////////////////////////////////////////////////////////////
// CTestDoc

IMPLEMENT_DYNCREATE(CTestDoc, CDocument)

BEGIN_MESSAGE_MAP(CTestDoc, CDocument)
    //{{AFX_MSG_MAP(CTestDoc)
    ON_COMMAND(ID_CHANNELS_C1FOREST, OnChannelsC1forest)
    ON_COMMAND(ID_CHANNELS_C1GARDEN, OnChannelsC1garden)
    ON_COMMAND(ID_CHANNELS_CASTLE1, OnChannelsCastle1)
    ON_COMMAND(ID_CHANNELS_CEMETARY, OnChannelsCemetery)
    ON_COMMAND(ID_CHANNELS_HOUSA, OnChannelsHousa)
    ON_COMMAND(ID_CHANNELS_HOUSB, OnChannelsHousb)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////
// CTestDoc construction/destruction

CTestDoc::CTestDoc()
{
    // TODO: add one-time construction code here
    m_bCurrentMIT = FALSE;
    m_bCurrentSIT = FALSE;
    m_strCurrentMIT.Empty();
    m_strCurrentSIT.Empty();
    m_nMITing = 0;
}

CTestDoc::~CTestDoc()
{
}

BOOL CTestDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)
    TRACE("CTestDoc::OnNewDocument\n");
}
```

TestDoc.cpp

```
//m_dataPP.Open(_T("ftp://88.1.26.2/clforest.sit"));

return TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CTestDoc serialization

void CTestDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CTestDoc diagnostics

#ifdef _DEBUG
void CTestDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CTestDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CTestDoc commands

void CTestDoc::OnCloseDocument()
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE("CTestDoc::OnCloseDocument\n");

    m_dataPPMIT.Close();

    CDocument::OnCloseDocument();
}

void CTestDoc::OnChannelsClforest()
{
    // TODO: Add your command handler code here
    if(!m_nMITing)
    {
        m_strCurrentMIT = "0010csin.sit";
    }
}
```

```

        CString URL(UNICHAT_FTP);
        URL += m_strCurrentMIT;
        m_bCurrentMIT = FALSE;
        m_nMITing = 1;
        m_dataPPMIT.Open(URL);
    }
    else
    {
        AfxMessageBox("Downloadng...");
    }
}

void CTestDoc::OnChannelsClgarden()
{
    // TODO: Add your command handler code here
    if(!m_nMITing)
    {
        m_strCurrentMIT = "0020csin.sit";

        CString URL(UNICHAT_FTP);
        URL += m_strCurrentMIT;
        m_bCurrentMIT = FALSE;
        m_nMITing = 1;
        m_dataPPMIT.Open(URL);
    }
    else
    {
        AfxMessageBox("Downloadng...");
    }
}

void CTestDoc::OnChannelsCastle1()
{
    // TODO: Add your command handler code here
    if(!m_nMITing)
    {
        m_strCurrentMIT = "0000csin.sit";

        CString URL(UNICHAT_FTP);
        URL += m_strCurrentMIT;
        m_bCurrentMIT = FALSE;
        m_nMITing = 1;
        m_dataPPMIT.Open(URL);
    }
    else
    {
        AfxMessageBox("Downloadng...");
    }
}

void CTestDoc::OnChannelsCemetary()
{
    // TODO: Add your command handler code here
    if(!m_nMITing)
    {
        m_strCurrentMIT = "1000csin.sit";
    }
}

```

```

        CString URL(UNICHAT_FTP);
        URL += m_strCurrentMIT;
        m_bCurrentMIT = FALSE;
        m_nMITing = 1;
        m_dataPPMIT.Open(URL);
    }
    else
    {
        AfxMessageBox("Downloadng...");
    }
}

void CTestDoc::OnChannelsHousa()
{
    // TODO: Add your command handler code here
    if(!m_nMITing)
    {
        m_strCurrentMIT = "2000csin.sit";

        CString URL(UNICHAT_FTP);
        URL += m_strCurrentMIT;
        m_bCurrentMIT = FALSE;
        m_nMITing = 1;
        m_dataPPMIT.Open(URL);
    }
    else
    {
        AfxMessageBox("Downloadng...");
    }
}

void CTestDoc::OnChannelsHousb()
{
    // TODO: Add your command handler code here
    if(!m_nMITing)
    {
        m_strCurrentMIT = "30000csin.sit";

        CString URL(UNICHAT_FTP);
        URL += m_strCurrentMIT;
        m_bCurrentMIT = FALSE;
        m_nMITing = 1;
        m_dataPPMIT.Open(URL);
    }
    else
    {
        AfxMessageBox("Downloadng...");
    }
}

void CTestDoc::SetCurrentMIT(BOOL available)
{
    m_bCurrentMIT = available;
}

void CTestDoc::SetCurrentSIT(BOOL available)

```

TestDoc.cpp

```
{
    m_bCurrentSIT = available;
}

void CTestDoc::OnStopBinding(WPARAM wParam, LPARAM lParam)
{
    TRACE("CTestDoc::OnStopBinding\n");

    m_bCurrentMIT = TRUE;
    m_nMITing = 0;
}
```


TestDoc.h

```
// TestDoc.h : interface of the CTestDoc class
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_TESTDOC_H__0BCFBAAC_B807_11D1_9169_444553540001__INCLUDED_)
#define AFX_TESTDOC_H__0BCFBAAC_B807_11D1_9169_444553540001__INCLUDED_

#include "JunAsyncMF.h" // Added by ClassView
#include "JunDataPathProp.h" // Added by ClassView
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CTestDoc : public CDocument
{
protected: // create from serialization only
    CTestDoc();
    DECLARE_DYNCREATE(CTestDoc)

// Attributes
public:
    CJunDataPathProp* GetJunDPPMIT() { return &m_dataPPMIT; }
    BOOL IsCurrentMITOK() { return m_bCurrentMIT;}

// Operations
public:

    int m_nMITing; // 1:MIT, 2:SIT, 0:Default

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CTestDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual void OnCloseDocument();
    //}AFX_VIRTUAL

// Implementation
public:
    void OnStopBinding(WPARAM wParam, LPARAM lParam);
    void SetCurrentMIT(BOOL available);
    void SetCurrentSIT(BOOL available);
    virtual ~CTestDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    BOOL m_bCurrentMIT;
    BOOL m_bCurrentSIT;
    CString m_strCurrentMIT;
```

TestDoc.h

```
CString m_strCurrentSIT;  
CJunDataPathProp m_dataPPMIT;  
CJunDataPathProp m_dataPPSIT;
```

```
//{{AFX_MSG(CTestDoc)  
afx_msg void OnChannelsClforest();  
afx_msg void OnChannelsClgarden();  
afx_msg void OnChannelsCastle1();  
afx_msg void OnChannelsCemetary();  
afx_msg void OnChannelsHousa();  
afx_msg void OnChannelsHousb();  
//}}AFX_MSG  
DECLARE_MESSAGE_MAP()
```

```
};
```

```
////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}  
// Microsoft Developer Studio will insert additional declarations immediately  
before the previous line.
```

```
#endif //
```

```
!defined(AFX_TESTDOC_H__0BCFBAAC_B807_11D1_9169_444553540001__INCLUDED_)
```

TestView.cpp

```
// TestView.cpp : implementation of the CTestView class
//

#include "stdafx.h"
#include "Test.h"

#include "TestDoc.h"
#include "TestView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CTestView

IMPLEMENT_DYNCREATE(CTestView, CFormView)

BEGIN_MESSAGE_MAP(CTestView, CFormView)
   //{{AFX_MSG_MAP(CTestView)
    ON_BN_CLICKED(IDC_GETSTATUS, OnGetstatus)
   //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////
// CTestView construction/destruction

CTestView::CTestView()
    : CFormView(CTestView::IDD)
{
    //{{AFX_DATA_INIT(CTestView)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    // TODO: add construction code here
}

CTestView::~CTestView()
{
}

void CTestView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTestView)
    // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BOOL CTestView::PreCreateWindow(CREATESTRUCT& cs)
{

```

TestView.cpp

```
// TODO: Modify the Window class or styles here by modifying
// the CREATESTRUCT cs

return CFormView::PreCreateWindow(cs);
}

////////////////////////////////////
// CTestView printing

BOOL CTestView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CTestView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CTestView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

void CTestView::OnPrint(CDC* pDC, CPrintInfo*)
{
    // TODO: add code to print the controls
}

////////////////////////////////////
// CTestView diagnostics

#ifdef _DEBUG
void CTestView::AssertValid() const
{
    CFormView::AssertValid();
}

void CTestView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CTestDoc* CTestView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTestDoc)));
    return (CTestDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////
// CTestView message handlers

void CTestView::OnGetstatus()
{
    // TODO: Add your control notification handler code here
}
```

TestView.cpp

```
CTestDoc* pDoc = GetDocument();
if(!pDoc->IsCurrentMITOK())
{
    AfxMessageBox("File Downloading...");
    return;
}

CJunDataPathProp* pDPP = GetDocument()->GetJunDPPMIT();

DWORD size = pDPP->GetLength();

char* buffer = new char[size];
pDPP->SeekToBegin();
pDPP->Read(buffer, size);

AfxMessageBox(buffer);

delete [] buffer;
}
```

TestView.h

```
// TestView.h : interface of the CTestView class
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_TESTVIEW_H__0BCFBAAE_B807_11D1_9169_444553540001__INCLUDED_)
#define AFX_TESTVIEW_H__0BCFBAAE_B807_11D1_9169_444553540001__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CTestView : public CFormView
{
protected: // create from serialization only
    CTestView();
    DECLARE_DYNCREATE(CTestView)

public:
    //{{AFX_DATA(CTestView)
    enum{ IDD = IDD_TEST_FORM };
    // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

    // Attributes
public:
    CTestDoc* GetDocument();

    // Operations
public:

    // Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTestView)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnPrint(CDC* pDC, CPrintInfo*);
    //}}AFX_VIRTUAL

    // Implementation
public:
    virtual ~CTestView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

    // Generated message map functions
protected:
    //{{AFX_MSG(CTestView)
    afx_msg void OnGetstatus();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
}
```

TestView.h

```
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifdef _DEBUG // debug version in TestView.cpp
inline CTestDoc* CTestView::GetDocument()
{ return (CTestDoc*)m_pDocument; }
#endif

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_TESTVIEW_H__0BCFBAAE_B807_11D1_9169_444553540001__INCLUDED_)
```

TileMap.cpp

```
// TileMap.cpp: implementation of the CTileMap class.
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT
//=====
/* =====
; map000
#VERSION=1.00;
#TILESIZE=(64,32);          //(64,32)
#SCREENSIZE=(608,384);
#ROW=(0);
{
// nResID=nCellID,nElev,wImOp,T.nEA,wSpriteType,nMSPT,T.wDA,strHyperlink;
// -1=;          // NULL Tile
// nResID=nCellID;          // ,DEFAULT_IO,0,SPRITE_TILE,DA_OPEN;
11=7,0,4096,0,1,0,15,0010csin;
...
}
#ROW=(1);
{
...
=====*/

#include "stdafx.h"
#include "TileMap.h"

#include "Parser.h"
#include "ResMan.h"
#include "UC2Ani/PhSprite.h"
#include "UC2Ani/OSBView.h"
#include "UC2Ani/SprItLst.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

extern CParser gParser;
extern CResMan gResMan;

double GetMapEditorVersion();

const char* MAPFILE_FILTER = "Map Information Tables(*.MIT)|*.MIT|"
                             "All files (*.*)|*.*||";
const char* MAPFILE_EXT = ".mit";          // should be lower case

////////////////////////////////////
// Construction/Destruction
////////////////////////////////////

CTileMap::CTileMap()
{
    TRACE0("CTileMap::CTileMap()\n");
    m_sScr          = CSize(0, 0);    // Screen Width, Height
}
```


TileMap.cpp

```

    m_sT      = CSize(0, 0);    // Tile width, height
    m_sTH     = CSize(0, 0);    // Half Tile width, height
    m_sTiles  = CSize(0, 0);
    m_apTile  = NULL;
    m_pSpriteList = NULL;
    m_pAniList = NULL;
    m_pPalette = NULL;
}

CTileMap::~CTileMap()
{
    TRACE0("CTileMap::~CTileMap()\n");
    DeleteTiles();
}

void CTileMap::DeleteTiles()
{
    if (!m_apTile)
        return;
    for (int ny=0; ny < m_sTiles.cy; ny++)
    {
        for (int nx=0; nx < m_sTiles.cx; nx++)
        {
            TILE& T = m_apTile[ny][nx];
            if (T.pPS)
            {
                RemoveList(T.pPS);    // Time-consuming
                delete T.pPS;
                T.pPS = NULL;
            }
#ifdef MAPEDITOR
            if (T.pGridPS)
                delete T.pGridPS;
            T.pGridPS = NULL;
#endif
        }
        delete [] m_apTile[ny];
        m_apTile[ny] = NULL;
    }
    delete [] m_apTile;
    m_apTile = NULL;
    m_strMapFile.Empty();
}

// Feb 5, 1998
// PROBLEM: Get the nearest diamond (tile) center for a given point
// Let  $a = w/2$ ,  $b = h/2$ 
//  $x/a + y/b = 2M$ ,  $M=0,1,2,\dots$  - (1)
//  $x/a - y/b = 2N$ ,  $N=\dots,-2,-1,0,1,2,\dots$  - (2)
//  $2M-1 < x/a + y/b \leq 2M+1$ 
//  $2N-1 < x/a - y/b \leq 2N+1$ 
// Let  $S=x/a + y/b$ ,  $D=x/a - y/b$ , Here  $S$  and  $D$  may not be an integer!
//  $(S-1)/2 \leq M < (S+1)/2$  - (3)
//  $(D-1)/2 \leq N < (D+1)/2$  - (4)
// So the problem reduces to finding such an integer that
// satisfies above inequalities.
// If we find such integers  $M$  and  $N$ , we can solve for the center point

```

```

// By adding and subtracting equations (1) and (2), we get
// C = (a(M+N), b(M-N)) - (5)
CPoint CTileMap::GetNearestTileCenter(const CPoint& pt) const
{
    float fx = float(pt.x)/m_sTH.cx;    // x/a
    float fy = float(pt.y)/m_sTH.cy;    // y/b
    double fM = (fx + fy + 1.)/2.;      // (S-1)/2 <= M < (S+1)/2
    double fN = (fx - fy + 1.)/2.;      // (D-1)/2 <= N < (D+1)/2 can be
negative
    if (fN <= 0.)    // Consider a problem to find an integer that
satisfies
        fN--;    // -1.5 <= N < -0.5, But int(-0.5)=0
    int M = int(fM);
    int N = int(fN);
    return CPoint(m_sTH.cx*(M + N), m_sTH.cy*(M - N));
}

CPoint CTileMap::GetNearestTileIndex(const CPoint& pt) const
{
    CPoint ptCtr(GetNearestTileCenter(pt));
    int ny = ptCtr.y / m_sTH.cy;
    int nx = (ny%2 == 0)
        ? ptCtr.x / m_sT.cx
        : (ptCtr.x / m_sTH.cx - 1) / 2;
    // Boundary conditions not to get any invalid indices
    if (nx >= m_sTiles.cx)
        nx = m_sTiles.cx - 1;
    if (nx < 0)
        nx = 0;
    if (ny >= m_sTiles.cy)
        ny = m_sTiles.cy - 1;
    if (ny < 0)
        ny = 0;
    return CPoint(nx, ny);
}

CPoint CTileMap::GetNearestTileIndex(CPhasedSprite* pPS) const
{
    CPoint point(pPS->GetCenter());
    point.y += pPS->GetElevation();    // Adjust for elevation!
    return GetNearestTileIndex(point);
}

// You must call SetTileSize() before calling this
BOOL CTileMap::Create(const int W, const int H)
{
    DeleteTiles();
    if ((m_sT.cx < 4) || (m_sT.cy < 2))
        SetTileSize(CSize(64, 32));
    m_sScr = CSize(W, H);
    m_sTiles = CSize(m_sScr.cx / m_sT.cx + 1, m_sScr.cy / m_sTH.cy + 1);
    TRACE("# of tiles (%d,%d) => %d\n", m_sTiles.cx, m_sTiles.cy,
m_sTiles.cx*m_sTiles.cy);
    m_apTile = new LPTILE[m_sTiles.cy];
    for (int ny=0; ny < m_sTiles.cy; ny++)
    {
        m_apTile[ny] = new TILE[m_sTiles.cx];
    }
}

```

```

        for (int nx=0; nx < m_sTiles.cx; nx++)
        {
            TILE& T = m_apTile[ny][nx];
            T.pPS = NULL;
#ifdef MAPEDITOR
            T.pGridPS = NULL;
#endif
            T.nEA = 0;
            T.wDA = DA_CLOSED;
        }
    }
    return TRUE;
}

// Synchronize actor elevation with sprite elevation
void CTileMap::SynchronizeEA(const CPoint& ptID)
{
    int nEA = GetPS(ptID) ? GetPS(ptID)->GetElevation() : 0;
    SetEA(ptID, nEA);
}

void CTileMap::IncreaseElevations(const int nPixelBy)
{
    for (int ny=0; ny < m_sTiles.cy; ny++)
    {
        for (int nx=0; nx < m_sTiles.cx; nx++)
        {
            TILE& T = m_apTile[ny][nx];
            if (T.pPS)
            {
                T.pPS->IncElevation(nPixelBy);
                T.pPS->MoveBy(0, -nPixelBy);
                T.pPS->SetZByEarth();
                T.nEA += nPixelBy;
            }
        }
    }
}

// Can the actor go further with his direction attribute?
BOOL CTileMap::GetActorNextTileID(CPoint& ptTID, const WORD wDA) const
{
    const int& x = ptTID.x;
    const int& y = ptTID.y;
    // Check if this tile has the same direction attribute with the actor
    if ((m_apTile[y][x].wDA != DA_CLOSED) && // we should escape!
        (m_apTile[y][x].wDA & wDA) != wDA)
        return FALSE;

    switch (wDA)
    {
        case DA_BL: ptTID = (y % 2) ? CPoint(x, y-1) : CPoint(x-1, y-1);
        break;
        case DA_BR: ptTID = (y % 2) ? CPoint(x+1, y-1) : CPoint(x, y-1);
        break;
        case DA_FR: ptTID = (y % 2) ? CPoint(x, y+1) : CPoint(x-1,
y+1); break;
    }
}

```

```

        case DA_FL: ptTID = (y % 2) ? CPoint(x+1, y+1) : CPoint(x, y+1);
        break;
    }
    if ((ptTID.x >= 0) && (ptTID.y >= 0) &&
        (ptTID.x < m_sTiles.cx) && (ptTID.y < m_sTiles.cy))
        return TRUE; // Valid tile id
    return FALSE;
}

// Load tiles info from text file
// Only for Version with 0.99 or below
BOOL CTileMap::Load(const char* path) // obsolete for Ver 1.0 or above
{
    if ((path == NULL) || (lstrlen(path) == 0))
    {
        ::SetCurrentDirectory(*gResMan.GetResPath());
        // Show an Open File dialog to get the name.
        CFileDialog dlg(TRUE, // Open
                        NULL, // No default extension
                        m_strMapFile, // Initial file name
                        OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
                        MAPFILE_FILTER);

        if (dlg.DoModal() == IDOK)
            m_strMapFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        m_strMapFile = path;
        gResMan.ExpandResName(m_strMapFile, MAPFILE_EXT);
    }

    CTextFileBuffer tfb(gParser.GetMaxBuffer());
    if (!tfb.Load(m_strMapFile))
    {
        return FALSE;
    }
    gResMan.ExpandResName(m_strMapFile, MAPFILE_EXT);

    double fVersion; // file version

    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (!gParser.SetLeftToken('#'))
            continue;
        CString strBuf;
        if (!gParser.GetValueRightToken(strBuf, '='))
            continue;

        //////////////////////////////////////
        if (lstrcmpi(strBuf, "VERSION") == 0)
        {

```

```

        gParser.SetLeftToken('=');
        gParser.GetValueRightToken(fVersion, ';');
        if (fVersion <= 0.90)
        {
            CString strMsg;
            strMsg.Format("MIT: Old Version %.2f", fVersion);
            AfxMessageBox(strMsg);
        }
    }
    else if (lstrcmpi(strBuf, "TILESIZE") == 0)
    {
        gParser.SetLeftToken('=');
        CSize sT;
        gParser.GetValueRightToken(sT, ';');
        SetTileSize(sT);
    }
    else if (lstrcmpi(strBuf, "SCREENSIZE") == 0)    // matching!
    {
        gParser.SetLeftToken('=');
        CSize sScr;
        gParser.GetValueRightToken(sScr, ';');
        Create(sScr);    // Create Tiles
    }
    else if (lstrcmpi(strBuf, "ROW") == 0)
    {
        gParser.SetLeftToken('(');
        int ny;
        gParser.GetValueRightToken(ny, ')');
        if (ny >= m_sTiles.cy)
        {
            TRACE("TILE Invalid row (%d)!\n", ny);
            continue;
        }
        if (!LoadRow(tfb, ny, fVersion))
            continue;
    }
    else
    {
        TRACE1("Unknown Data type in Map file(%s)!\n", strBuf);
    }
}

#ifdef MAPEDITOR
    LoadGrid();
#endif
return TRUE;
}

// Load each tile row from memory buffer
// nResID=nCellID,nElev,wImOp,T.nEA,wType,nMSPT,T.wDA,strHyperlink;
// -1=;    // NULL Tile
// nResID=nCellID;    // ,DEFAULT_IO,0,SPRITE_TILE,DA_OPEN;
BOOL CTileMap::LoadRow(CTextFileBuffer& tfb, const int ny, const double
fVersion)
{
    int nx=0;
    if (ny >= m_sTiles.cy)
    {

```

```

        TRACE("TILE Invalid row (%d)!\n", ny);
        return FALSE;
    }
    while (tfb.ReadString())
    {
        gParser.CopyBuffer(tfb.GetString());
        if (gParser.IsCommentLine())
            continue;
        if (gParser.SetLeftToken('{'))
        {
            if (gParser.SetLeftToken('}'))
                break;
            continue;
        }
        if (gParser.SetLeftToken(' '))
            break;

        if (nx >= m_sTiles.cx)
        {
            TRACE("TILE Invalid column (%d)!\n", nx);
            continue;
        }
        CString strResName;
        TILE& T = m_apTile[ny][nx];
        ASSERT(!T.pPS); // already initialized as a NULL
        ASSERT(T.nEA == 0);
        // ASSERT(T.wDA == DA_CLOSED);
        int nCellID;
        int nElev = 0;
        WORD wImOp = DEFAULT_IO;
        WORD wType = SPRITE_TILE;
        int nMSPT = 0;
        CString strHyperlink;

        /* if (fVersion <= 0.90) // Version 0.90 (before Mar 7, '98)
        {
            gParser.GetValueRightToken(strResName, '=');
            if (strResName[0] == '0') // NULL tile
            {
                T.wDA = DA_CLOSED;
                nx++;
                continue;
            }
            gResMan.PrefixUDS00(strResName);
        }
        else // New Version
        {
            int nResID;
            gParser.GetValueRightToken(nResID, '=');
            if (nResID < 0) // -1 for NULL Tile
            {
                if (fVersion < 1.01)
                {
                    T.wDA = DA_CLOSED;
                }
                else // V1.01

```

```

        {
            if (!gParser.GetValueRightToken(T.nEA, ',',
';'))
                T.nEA = nElev;
            else if (!gParser.GetValueRightToken(T.wDA,
',', ';'))
                T.wDA = DA_CLOSED;        // default for
NULL tile
        }
        nx++;
        continue;
    }
    CString* pStr = gResMan.GetSpriteName(nResID);
    if (!pStr)
    {
        strResName.Format("ResID=%d in Map not found!",
nResID);
        AfxMessageBox(strResName);
        continue;
    }
    strResName = *pStr;
//    }

    T.nEA = nElev;        // default
    T.wDA = DA_OPEN;    // default condition

    gParser.GetValueRightToken(nCellID, ',', ';');
    if (!gParser.GetValueRightToken(nElev, ',', ';'))
        nElev = 0;
    else // continue to the next delimiter
    {
        if (!gParser.GetValueRightToken(wImOp, ',', ';'))
            wImOp = DEFAULT_IO;
        else
        {
            if (!gParser.GetValueRightToken(T.nEA, ',', ';'))
                T.nEA = nElev;
            else
            {
                if (!gParser.GetValueRightToken(wType, ',',
';'))
                    wType = SPRITE_TILE;
                else
                {
                    if (!gParser.GetValueRightToken(nMSPT,
',', ';'))
                        nMSPT = 0;
                    else
                    {
                        if
(!gParser.GetValueRightToken(T.wDA, ',', ';'))
                            T.wDA = DA_OPEN;
                        else
                        {
                            if
(!gParser.GetValueRightToken(strHyperlink, ',', ';'))
                                strHyperlink.Empty();

```

```

    }
    }
    }
    }
}

// Resource Allocation via Resource Manager
CPhasedSprite* pPS = gResMan.LoadPhasedSprite(strResName);
if (!pPS)
{
    strResName += " not found!";
    AfxMessageBox(strResName);
    continue;
}

pPS->SetCell(nCellID);
CPoint ptLT(GetLT(nx, ny));
if (nElev)
{
    pPS->SetElevation(nElev);
    ptLT.y -= nElev;
}
pPS->SetLT(ptLT);
pPS->SetType(wType);
if (fVersion < 1.01)
{
    wImOp &= ~NO_COLORKEY;
    if (nMSPT && !pPS->GetAniType())
        nMSPT = 0;
}
pPS->SetImOp(wImOp);
pPS->SetZByEarth();
if (nMSPT > 0)
    pPS->SetMSPT(nMSPT);
if (!strHyperlink.IsEmpty())
    pPS->SetHyperlink(strHyperlink);
T.pPS = pPS;

InsertList(pPS);
if (m_pPalette)
    pPS->MapColorsToPalette(m_pPalette);
nx++;
}
return TRUE;
}

BOOL CTileMap::Save(CStdioFile& f)
{
    CString str;
    str.Format("\n#TILESIZE=(%d,%d);\n", m_sT.cx, m_sT.cy);    // (64,32)
    f.WriteString(str);

    str.Format("#SCREENSIZE=(%d,%d);\n", m_sScr.cx, m_sScr.cy);
    f.WriteString(str);

    int n=0;

```



```

for (int ny=0; ny < m_sTiles.cy; ny++)
{
    str.Format("#ROW=(%d);\n{\n", ny);
    f.WriteString(str);
    for (int nx=0; nx < m_sTiles.cx; nx++)
    {
        TILE& T = m_apTile[ny][nx];
        if (T.pPS)
        {
            CString strResName(*T.pPS->GetDIB()->GetName());
            gResMan.MakeResName(strResName);
            str.Format("%s=%d,%d,%d,%d,%d,%d;\n",
// strResName, // #VERSION=0.90
                int nResID = gResMan.GetSpriteID(strResName);
                if (nResID < 0)
                {
                    strResName += " not found";
                    AfxMessageBox(strResName);
                }
                if ((T.pPS->GetElevation() == 0) &&
                    (T.pPS->GetImOp() == DEFAULT_IO) &&
                    (T.nEA == 0) &&
                    (T.pPS->GetType() == SPRITE_TILE) &&
                    (T.wDA == DA_OPEN) &&
                    (!T.pPS->HasHyperlink()))
                {
                    // typical case
                    str.Format("%d=%d", nResID, T.pPS-
>GetCellID());
                }
                else
                {
                    str.Format("%d=%d,%d,%d,%d,%d,%d,%d", nResID,
                        T.pPS->GetCellID(),
                        T.pPS->GetElevation(),
                        T.pPS->GetImOp(), T.nEA,
                        T.pPS->GetType(), T.pPS->GetMSPT(),
                        T.wDA);
                    if (T.pPS->HasHyperlink())
                    {
                        str += ',';
                        str += *T.pPS->GetHyperlink();
                    }
                }
                str += ";\n";
            }
        }
        else // NULL Tile, V1.01
        {
            str.Format("-1=%d,%d;\n", T.nEA, T.wDA);
        }
        f.WriteString(str);
        n++;
    }
    f.WriteString("}\n");
}
str.Format("; %d tiles.\n", n);
f.WriteString(str);
// str.Format("%s: %d sprites written.", m_strMapFile, n);

```

TileMap.cpp

```

//      AfxMessageBox(str);
//      return TRUE;
}

void CTileMap::DrawHyperlink(CDC* pDC, const CPoint& ptC, const int nType)
{
    CPen penRed(PS_SOLID, 1, (nType == CPhasedSprite::HLINK_U2_EXIT) ?
        PALETTERGB(255,0,0) :
        PALETTERGB(0,255,0));
    CBrush brush((nType == CPhasedSprite::HLINK_U2_EXIT) ?
        PALETTERGB(255,0,0) :
        PALETTERGB(255,255,0)); // Yellow HS_CROSS
    CPen* penOld = pDC->SelectObject(&penRed);
    CBrush* brOld = pDC->SelectObject(&brush);
    pDC->SetBkMode(TRANSPARENT);

    CRect rc(ptC.x-5, ptC.y-5, ptC.x+5, ptC.y+5);
    pDC->Ellipse(rc);

    pDC->SelectObject(brOld);
    pDC->SelectObject(penOld);
}

// ptC0 / ptC1 ` ptC3 (draw next time)
//      ` ptC2 /
void CTileMap::DrawElevationGraph(CDC* pDC)
{
    CRect rcDraw;
    // OnPrepareDC(pDC); // CScroll changes the viewport origin and mapping
    mode.
    pDC->GetClipBox(rcDraw);

    CPen penRed(PS_SOLID, 1, PALETTERGB(255,0,0));
    CPen penRedDot(PS_DOT, 1, PALETTERGB(255,0,0));
    CPen penGreen(PS_SOLID, 1, PALETTERGB(0,255,0));
    CPen penGreenDot(PS_DOT, 1, PALETTERGB(0,255,0));
    // CBrush brush(HS_CROSS, PALETTERGB(255,255,0)); // Yellow

    CPen* penOld = pDC->SelectObject(&penRed);
    pDC->SetBkMode(TRANSPARENT);
    // CBrush* brOld = pDC->SelectObject(&brush);

    // pDC->Rectangle(&rcDraw);
    for (int ny=0; ny < m_sTiles.cy; ny += 2)
    {
        int nx2 = m_sTiles.cx - 1;
        for (int nx=0; nx < nx2; nx++)
        {
            TILE &T0 = m_apTile[ny][nx];
            CPoint ptC0(GetCenter(nx, ny));
            if (T0.pPS && T0.pPS->GetElevation())
            {
                CPoint ptC00(ptC0); // Save before elevation
                ptC0.y -= T0.pPS->GetElevation();
                pDC->MoveTo(ptC00);
                pDC->SelectObject((T0.pPS->GetElevation() < 0) ?
                    &penRed : &penRedDot);
            }
        }
    }
}

```

```

        pDC->LineTo(ptC0);
    }
    if (T0.pPS && T0.pPS->HasHyperlink())
        DrawHyperlink(pDC, ptC0, T0.pPS->GetLinkType());

    TILE &T3 = m_apTile[ny][nx+1];
    CPoint ptC3(GetCenter(nx+1, ny));
    if (T3.pPS && T3.pPS->GetElevation())
    {
        ptC3.y -= T3.pPS->GetElevation();
    }

    if (ny >= 1)
    {
        TILE &T1 = m_apTile[ny-1][nx];          // upper line
        CPoint ptC1(GetCenter(nx, ny-1));
        if (T1.pPS && T1.pPS->GetElevation())
        {
            CPoint ptC10(ptC1);          // Save before
            ptC1.y -= T1.pPS->GetElevation();
            pDC->MoveTo(ptC10);
            pDC->SelectObject((T1.pPS->GetElevation() < 0)
                ? &penRed : &penRedDot);
            pDC->LineTo(ptC1);
        }
        if (T1.pPS && T1.pPS->HasHyperlink())
            DrawHyperlink(pDC, ptC1, T1.pPS->GetLinkType());

        pDC->MoveTo(ptC0);
        pDC->SelectObject(T1.pPS ? &penGreen : &penGreenDot);
        pDC->LineTo(ptC1);
        pDC->SelectObject(T3.pPS ? &penGreen : &penGreenDot);
        pDC->LineTo(ptC3);
    }
    if (ny < (m_sTiles.cy-1))
    {
        TILE &T2 = m_apTile[ny+1][nx];          // lower line
        CPoint ptC2(GetCenter(nx, ny+1));
        if (T2.pPS && T2.pPS->GetElevation())
        {
            CPoint ptC20(ptC2);          // Save before
            ptC2.y -= T2.pPS->GetElevation();
            pDC->MoveTo(ptC20);
            pDC->SelectObject((T2.pPS->GetElevation() < 0)
                ? &penRed : &penRedDot);
            pDC->LineTo(ptC2);
        }
        if (T2.pPS && T2.pPS->HasHyperlink())
            DrawHyperlink(pDC, ptC2, T2.pPS->GetLinkType());

        pDC->MoveTo(ptC0);
        pDC->SelectObject(T2.pPS ? &penGreen : &penGreenDot);
        pDC->LineTo(ptC2);
        pDC->SelectObject(T3.pPS ? &penGreen : &penGreenDot);
        pDC->LineTo(ptC3);
    }

```

```

    }
/*
    CPoint ptC(GetCenter(nx, ny));
    pDC->LineTo(ptC.x + m_sTH.cx, ptC.y + m_sTH.cy);
    pDC->LineTo(ptC.x + m_sT.cx, ptC.y);
    pDC->MoveTo(ptC.x, ptC.y);
    pDC->LineTo(ptC.x + m_sTH.cx, ptC.y - m_sTH.cy);
    pDC->LineTo(ptC.x + m_sT.cx, ptC.y);
*/
    }
    }
    pDC->SelectObject(penOld);
//    pDC->SelectObject(brOld);
}

void CTileMap::DrawActorElevationGraph(CDC* pDC)
{
    CRect rcDraw;
//    OnPrepareDC(pDC); // CScroll changes the viewport origin and mapping
mode.
    pDC->GetClipBox(rcDraw);

    CPen penRed(PS_SOLID, 1, PALETTERGB(255,0,0));
    CPen penRedDot(PS_DOT, 1, PALETTERGB(255,0,0));
    CPen penBlue(PS_SOLID, 1, PALETTERGB(0,0,255));
    CPen penBlueDot(PS_DOT, 1, PALETTERGB(0,0,255));
//    CBrush brush(HS_CROSS, PALETTERGB(255,255,0)); // Yellow

    CPen* penOld = pDC->SelectObject(&penRed);
    pDC->SetBkMode(TRANSPARENT);
//    CBrush* brOld = pDC->SelectObject(&brush);

//    pDC->Rectangle(&rcDraw);
    for (int ny=0; ny < m_sTiles.cy; ny += 2)
    {
        int nx2 = m_sTiles.cx - 1;
        for (int nx=0; nx < nx2; nx++)
        {
            TILE &T0 = m_apTile[ny][nx];
            CPoint ptC0(GetCenter(nx, ny));
            if (T0.nEA)
            {
                CPoint ptC00(ptC0); // Save before elevation
                ptC0.y -= T0.nEA;
                pDC->MoveTo(ptC00);
                pDC->SelectObject((T0.nEA < 0) ? &penRed :
&penRedDot);
                pDC->LineTo(ptC0);
            }

            TILE &T3 = m_apTile[ny][nx+1];
            CPoint ptC3(GetCenter(nx+1, ny));
            if (T3.nEA)
            {
                ptC3.y -= T3.nEA;
            }
        }
    }
}

```

```

if (ny >= 1)
{
    TILE &T1 = m_apTile[nx-1][ny-1];
    CPoint ptC1(GetCenter(nx, ny-1));
    if (T1.nEA)
    {
        CPoint ptC10(ptC1);    // Save before
        elevation
        ptC1.y -= T1.nEA;
        pDC->MoveTo(ptC10);
        pDC->SelectObject((T1.nEA < 0) ? &penRed :
        &penRedDot);

        pDC->LineTo(ptC1);
    }
    pDC->MoveTo(ptC0);
    pDC->SelectObject(T1.pPS ? &penBlue : &penBlueDot);
    pDC->LineTo(ptC1);
    pDC->SelectObject(T3.pPS ? &penBlue : &penBlueDot);
    pDC->LineTo(ptC3);
}
if (ny < (m_sTiles.cy-1))
{
    TILE &T2 = m_apTile[nx][ny+1];
    CPoint ptC2(GetCenter(nx, ny+1));
    if (T2.nEA)
    {
        CPoint ptC20(ptC2);    // Save before
        elevation
        ptC2.y -= T2.nEA;
        pDC->MoveTo(ptC20);
        pDC->SelectObject((T2.nEA < 0) ? &penRed :
        &penRedDot);

        pDC->LineTo(ptC2);
    }
    pDC->MoveTo(ptC0);
    pDC->SelectObject(T2.pPS ? &penBlue : &penBlueDot);
    pDC->LineTo(ptC2);
    pDC->SelectObject(T3.pPS ? &penBlue : &penBlueDot);
    pDC->LineTo(ptC3);
}

/*
CPoint ptC(GetCenter(nx, ny));
pDC->LineTo(ptC.x + m_sTH.cx, ptC.y + m_sTH.cy);
pDC->LineTo(ptC.x + m_sT.cx, ptC.y);
pDC->MoveTo(ptC.x, ptC.y);
pDC->LineTo(ptC.x + m_sTH.cx, ptC.y - m_sTH.cy);
pDC->LineTo(ptC.x + m_sT.cx, ptC.y);
*/

}
}
pDC->SelectObject(penOld);
// pDC->SelectObject(brOld);
}

// Synchronize EA (Actor Elevation) with Sprite Elevation
BOOL CTileMap::SynchronizeAllEA()
{

```

```

    for (int ny=0; ny < m_sTiles.cy; ny++)
    {
        for (int nx=0; nx < m_sTiles.cx; nx++)
        {
            TILE &T = m_apTile[ny][nx];
            T.nEA = T.pPS ? T.pPS->GetElevation() : 0;
        }
    }
    return TRUE;
}

/*
void CTileMap::RenderGrid(const HBITMAP hbm, const CRect* pClipRect)
{
    CRect rcDraw;
    rcDraw = (!pClipRect)
        ? CRect(0, 0, m_sScr.cx-1, m_sScr.cy-1)
        : *pClipRect;

    CDC dcMem;
    dcMem.CreateCompatibleDC(NULL);    // create a memory dc that is
compatible with current screen
    CBitmap* pbmOld = dcMem.SelectObject(CBitmap::FromHandle(hbm));

    CPen penOdd(PS_SOLID, 1, PALETTERGB(255,0,0));
    CPen penEven(PS_SOLID, 1, PALETTERGB(0,0,255));

    CPen* penOld = (CPen*)dcMem.SelectObject(&penOdd);

    int ny = rcDraw.top / m_sTH.cy;
    int ny2 = rcDraw.bottom / m_sTH.cy + 2;
    if (ny2 > m_sTiles.cy)
        ny2 = m_sTiles.cy;
    int nx1 = rcDraw.left / m_sT.cx;
    int nx2 = (m_sTH.cx + rcDraw.right) / m_sT.cx + 1;
    if (nx2 > m_sTiles.cx)
        nx2 = m_sTiles.cx;
    for (; ny < ny2; ny++)
    {
        CPoint ptLT;
        ptLT.y = GetLTy(ny);
        dcMem.SelectObject((ny % 2 == 0) ? &penEven : &penOdd);
        for (int nx = nx1; nx < nx2; nx++)
        {
            ptLT.x = GetLTx(nx, ny);
            int x1 = ptLT.x + m_sTH.cx - 2;
            int x2 = x1 + 1;
            int x3 = ptLT.x + m_sT.cx - 3;
            int y1 = ptLT.y + m_sTH.cy - 1;
            int y2 = y1 + 1;
            int y3 = ptLT.y + m_sT.cy - 1;
            dcMem.MoveTo(ptLT.x, y1);
            dcMem.LineTo(x1, ptLT.y);
            dcMem.MoveTo(x2, ptLT.y);
            dcMem.LineTo(x3, y1);
            dcMem.MoveTo(x3, y2);
            dcMem.LineTo(x2, y3);
        }
    }
}

```

```

        dcMem.MoveTo(x1, y3);
        dcMem.LineTo(ptLT.x, y2);
    }
}
dcMem.SelectObject(penOld);
dcMem.SelectObject(pbmOld);
}
*/

////////////////////////////////////
// Tile Manipulations

CPhasedSprite* CTileMap::HitTest(const CPoint& point)
{
    CPoint ptId(GetNearestTileIndex(point));
    return m_apTile[ptId.y][ptId.x].pPS;
}

BOOL CTileMap::InsideTile(const CPoint& ptTileID, const CPoint& point) const
{
    CPoint pt(point);
    CPoint ptLT(GetLT(ptTileID.x, ptTileID.y));
    if ((pt.x < ptLT.x) || (pt.x >= (ptLT.x+m_sT.cx)))
        return FALSE;
    TILE& T = m_apTile[ptTileID.y][ptTileID.x];
    if (T.pPS && T.pPS->GetElevation())
        pt.y -= T.pPS->GetElevation();
    if ((pt.y < ptLT.y) || (pt.y > (ptLT.y+m_sT.cy)))
        return FALSE;
    CPoint ptID(GetNearestTileIndex(pt));
    return (ptID == ptTileID);
}

// Sprite elevation
// returns 0 for NULL tile
int CTileMap::GetElevation(const CPoint& ptID) const
{
    TILE& T = m_apTile[ptID.y][ptID.x];
    return T.pPS ? T.pPS->GetElevation() : 0;
}

void CTileMap::InsertList(CPhasedSprite* pPS)
{
    if (!pPS)
        return;
    if (m_pSpriteList)
        m_pSpriteList->Insert(pPS);
    if (m_pAniList && pPS->GetAniType())
        m_pAniList->Insert(pPS);
}

void CTileMap::RemoveList(CPhasedSprite* pPS)
{
    if (!pPS)
        return;
    if (m_pAniList) // && pPS->GetAniType()
        m_pAniList->Remove(pPS);
}

```

```

        if (m_pSpriteList)
            m_pSpriteList->Remove(pPS);
    }

    // Find the entry tile matching szFromID
    // If not found, return any other entry point found last
    CPoint CTileMap::GetEntryID(LPCWSTR szFromID) const
    {
        CPoint ptID(10/2, 24/2);
        for (int ny=0; ny < m_sTiles.cy; ny++)
        {
            for (int nx=0; nx < m_sTiles.cx; nx++)
            {
                CPhasedSprite*& pPS = m_apTile[ny][nx].pPS;
                if (pPS && (pPS->GetLinkType() ==
CPhasedSprite::HLINK_U2_ENTRY))
                {
                    ptID = CPoint(nx, ny);
                    char szName[256];
                    lstrcpy(szName, *pPS->GetHyperlink());
                    ASSERT((szName[0] == 'e') && (szName[1] == ':'));
                    if (lstrcmpi(&szName[2], szFromID) == 0) // found
                    {
                        TRACE("Entry(From %s) found.\n", szFromID);
                        return ptID;
                    }
                }
            }
        }
        TRACE("Entry(From %s) not found!\n", szFromID);
        return ptID; // Not found, but anyway return another entry
    }

    //////////////////////////////////////
#ifdef MAPEDITOR

    BOOL CTileMap::LoadGrid(const WORD wResId)
    {
        BOOL bResult = TRUE;
        for (int ny=0; ny < m_sTiles.cy; ny++)
        {
            for (int nx=0; nx < m_sTiles.cx; nx++)
            {
                TILE &T = m_apTile[ny][nx];
                // Resource Allocation via Resource Manager
                CDIB* pDIB = gResMan.LoadDIB(wResId, TRUE); //
LoadMasterPalette
                if (!pDIB)
                {
                    TRACE("Grid resource load error!\n");
                    if (T.pGridPS)
                        delete T.pGridPS;
                    T.pGridPS = NULL;
                    bResult = FALSE;
                }
                else // DIB found
                {

```


TileMap.cpp

```

        CPhasedSprite* pPS = new CPhasedSprite;
        pPS->SetDIB(pDIB);          // link DIB resource to this
sprite
        if (T.pGridPS)
            delete T.pGridPS;
        T.pGridPS = pPS;

        pPS->SetType(SPRITE_TILE);
        pPS->SetNumCells(8, 1);

        CPoint ptID(nx, ny);
        SetGridImage(ptID, T.wDA);

        CPoint ptLT(GetLT(nx, ny));
        pPS->SetLT(ptLT);
        pPS->SetZByEarth();
        if (m_pPalette)
            pPS->MapColorsToPalette(m_pPalette);
    }
}
return bResult;
}

// Recalculate all DAs from the left top tile
BOOL CTileMap::RecalculateDA()
{
    for (int ny=0; ny < m_sTiles.cy; ny++)
    {
        for (int nx=0; nx < m_sTiles.cx; nx++)
        {
            TILE &T = m_apTile[ny][nx];
            CPoint ptID(nx, ny);
            SetDA(ptID, T.pPS ? T.wDA : DA_CLOSED);
        }
    }
    return TRUE;
}

// Overwrites existing tile
BOOL CTileMap::Insert(CPhasedSprite* pPS)
{
    if (!pPS)
        return FALSE;
    CPoint ptID(GetNearestTileIndex(pPS));
    TILE& T = m_apTile[ptID.y][ptID.x];
    if (T.pPS)
    {
        RemoveList(T.pPS);
        delete T.pPS;          // Replace existing sprite
    }
    CPoint ptLT(GetLT(ptID.x, ptID.y));
    ptLT.y -= pPS->GetElevation();
    pPS->SetLT(ptLT);
    pPS->SetZByEarth();
    T.pPS      = pPS;          // Newly assign the tile sprite
    T.nEA      = pPS->GetElevation();
}

```

```

        SetDA(ptID, DA_OPEN);
        InsertList(pPS);
        return TRUE;
    }

void CTileMap::Delete(CPhasedSprite* pPS)
{
    if (!pPS)
        return;
    CPoint ptID(GetNearestTileIndex(pPS));
    TILE& T = m_apTile[ptID.y][ptID.x];
    ASSERT(T.pPS == pPS);
    if (T.pPS)
    {
        RemoveList(T.pPS);
        delete T.pPS;
        T.pPS = NULL;
        T.nEA = 0;
        SetDA(ptID, DA_CLOSED);
    }
    return;
}

BOOL CTileMap::MoveSpriteTo(CPhasedSprite* pPS, CPoint& ptTo)
{
    if (!pPS)
        return FALSE;
    // Get the Sprite's Center point
    CPoint ptIdFrom(GetNearestTileIndex(pPS));
    CPoint ptIdTo(GetNearestTileIndex(ptTo));
    if (ptIdFrom == ptIdTo) // Same Indices
        return FALSE;

    TILE& TFrom = m_apTile[ptIdFrom.y][ptIdFrom.x];
    TILE& TTo = m_apTile[ptIdTo.y][ptIdTo.x];
    ASSERT(TFrom.pPS == pPS);

    if (TTo.pPS)
    {
        RemoveList(TTo.pPS);
        delete TTo.pPS; // Delete Destination Sprite
    }
    TTo.pPS = TFrom.pPS; // Newly assign the tile sprite
    TTo.nEA = TFrom.nEA;
    ASSERT(TFrom.pPS);

    TFrom.pPS = NULL; // Do not delete TFrom.pPS!
    TFrom.nEA = 0;
    TFrom.wDA = DA_OPEN;

    CPoint ptLT = GetLT(ptIdTo.x, ptIdTo.y);
    ptLT.y -= pPS->GetElevation();
    pPS->SetLT(ptLT);
    pPS->SetZByEarth();
    return TRUE;
}

```

```
// Set DA of the specified tile and other four tiles in its neighbor
void CTileMap::SetDA(const CPoint& ptID, const WORD wDA)
{
    const int& x = ptID.x;
    const int& y = ptID.y;
    m_apTile[y][x].wDA = wDA;
    SetGridImage(ptID, wDA);
    if ((y % 2) == 0) // y = 2n
    {
        if (y > 0)
        {
            if (x > 0)
            {
                WORD& wBL = m_apTile[y-1][x-1].wDA;
                if ((wDA & DA_BL) == DA_BL)
                    wBL |= DA_FL;
                else
                    wBL &= ~DA_FL;
                SetGridImage(CPoint(x-1, y-1), wBL);
            }
            WORD& wBR = m_apTile[y-1][x].wDA;
            if ((wDA & DA_BR) == DA_BR)
                wBR |= DA_FR;
            else
                wBR &= ~DA_FR;
            SetGridImage(CPoint(x, y-1), wBR);
        }
        if ((y+1) < m_sTiles.cy)
        {
            if (x > 0)
            {
                WORD& wFR = m_apTile[y+1][x-1].wDA;
                if ((wDA & DA_FR) == DA_FR)
                    wFR |= DA_BR;
                else
                    wFR &= ~DA_BR;
                SetGridImage(CPoint(x-1, y+1), wFR);
            }
            WORD& wFL = m_apTile[y+1][x].wDA;
            if ((wDA & DA_FL) == DA_FL)
                wFL |= DA_BL;
            else
                wFL &= ~DA_BL;
            SetGridImage(CPoint(x, y+1), wFL);
        }
    }
    else // y = 2n + 1
    {
        if (y > 0)
        {
            WORD& wBL = m_apTile[y-1][x].wDA;
            if ((wDA & DA_BL) == DA_BL)
                wBL |= DA_FL;
            else
                wBL &= ~DA_FL;
            SetGridImage(CPoint(x, y-1), wBL);
            if ((x+1) < m_sTiles.cx)

```

```

        {
            WORD& wBR = m_apTile[y-1][x+1].wDA;
            if ((wDA & DA_BR) == DA_BR)
                wBR |= DA_FR;
            else
                wBR &= ~DA_FR;
            SetGridImage(CPoint(x+1, y-1), wBR);
        }
    }
    if ((y+1) < m_sTiles.cy)
    {
        WORD& wFR = m_apTile[y+1][x].wDA;
        if ((wDA & DA_FR) == DA_FR)
            wFR |= DA_BR;
        else
            wFR &= ~DA_BR;
        SetGridImage(CPoint(x, y+1), wFR);
        if ((x+1) < m_sTiles.cx)
        {
            WORD& wFL = m_apTile[y+1][x+1].wDA;
            if ((wDA & DA_FL) == DA_FL)
                wFL |= DA_BL;
            else
                wFL &= ~DA_BL;
            SetGridImage(CPoint(x+1, y+1), wFL);
        }
    }
}

void CTileMap::SetGridImage(const CPoint& ptID, const WORD wDA)
{
    TILE& T = m_apTile[ptID.y][ptID.x];
    ASSERT(T.pGridPS);
    int nCellID=0;
    WORD wImOp = T.pGridPS->GetImOp() & ~ORIENT_MASK;
    switch (wDA)
    {
        case DA_OPEN:      nCellID = 1 + ptID.y % 2;      break;
        case DA_CLOSED:    nCellID = 0;                  break;
        case DA_FR:        nCellID = 3;                  break;
        case DA_FL:        nCellID = 3;                  wImOp |= IMAGE_FLIP;      break;
        case DA_BR:        nCellID = 3;                  wImOp |= (IMAGE_FLIP |
IMAGE_VERTICAL); break;
        case DA_BL:        nCellID = 3;                  wImOp |= IMAGE_VERTICAL;
break;
        case (DA_FR | DA_FL): nCellID = 4;              break;
        case (DA_BR | DA_BL): nCellID = 4;              wImOp |= IMAGE_VERTICAL;
break;
        case (DA_FR | DA_BL): nCellID = 5;              break;
        case (DA_FL | DA_BR): nCellID = 5;              wImOp |= IMAGE_FLIP;
break;
        case (DA_FR | DA_BR): nCellID = 6;              break;
        case (DA_FL | DA_BL): nCellID = 6;              wImOp |= IMAGE_FLIP;
break;
        case (DA_FR | DA_FL | DA_BR): nCellID = 7;      break;
    }
}

```

```

        case (DA_FR | DA_FL | DA_BL): nCellID = 7;        wImOp |= IMAGE_FLIP;
        break;
        case (DA_FR | DA_BR | DA_BL): nCellID = 7;        wImOp |= (IMAGE_FLIP |
IMAGE_VERTICAL); break;
        case (DA_FL | DA_BR | DA_BL): nCellID = 7;        wImOp |=
IMAGE_VERTICAL; break;
        default:      TRACE("Invalid DA: %0x\n", wDA);
        }
        T.pGridPS->SetCell(nCellID);
        T.pGridPS->SetImOp(wImOp);
    }

// Render the tiles to the given DIB within specified region
// I'll not use this method.
// Tiles will be rendered as normal sprites in the CSpriteList.
// Grid tiles will use this render method
void CTileMap::RenderGrid(CDIB* pDIB, const CRect* pClipRect)
{
    CRect rcDraw;
    if (!pClipRect)          // pClipRect == NULL means drawing on the
entire area
        pDIB->GetRect(rcDraw);
    else
        rcDraw = *pClipRect;

    int ny = 0; // = rcDraw.top / m_sTH.cy;
    int ny2 = m_sTiles.cy; // = rcDraw.bottom / m_sTH.cy + 2;
    if (ny2 > m_sTiles.cy)
        ny2 = m_sTiles.cy;
    int nx1 = rcDraw.left / m_sT.cx;
    int nx2 = (m_sTH.cx + rcDraw.right) / m_sT.cx + 1;
    if (nx2 > m_sTiles.cx)
        nx2 = m_sTiles.cx;

    for (; ny < ny2; ny++)
    {
        for (int nx=nx1; nx < nx2; nx++)
        {
            TILE& T = m_apTile[ny][nx];
            if (T.pGridPS)
                T.pGridPS->Render(pDIB, &rcDraw);
        }
    }
}

#endif // MAPEDITOR

```

TileMap.h

```
// TileMap.h: interface for the CTileMap class.
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT NETWORK INC
//=====

#if !defined(AFX_TILEMAP_H_D5010EC6_A1F9_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_TILEMAP_H_D5010EC6_A1F9_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifdef MAPEDITOR
#include "MapEd/resource.h"    // DIB_GRID_64X32
#else
#include "resource.h"
#endif

enum ELEVATION_UNITS
{
    ELEVATION_UNIT        = 8,
    ELEVATION_DIFF        = 16, // Max Elevation an actor can go on it
    ELEVATION_LIMIT       = 999
};

class CDIB;
class CPhasedSprite;
class CSpriteList;
class CTextFileBuffer;

class CTileMap : public CObject
{
public:
    CTileMap();
    virtual ~CTileMap();

    typedef struct
    {
        CPhasedSprite* pPS;    // includes nCellID, nImOp
#ifdef MAPEDITOR
        CPhasedSprite* pGridPS; // for Grid Cell
#endif
        int            nEA;    // Elevation for Actors
        WORD   wDA;    // Direction Attributes
        // int            nResID;    // DIB Resource ID in ResMan
        // int            nCellID;
        // int            nElev;    // Elevation, z = f(x,y)
        // int            nImOp;    // IMAGE_OPERATION
    } TILE;
    IMAGE_VERTICAL
    typedef TILE* LPTILE;

// Attributes
    CSize      GetTileSize() const    { return m_sT; }

```

TileMap.h

```

CSize      GetHalfTileSize() const { return m_sTH; }
CSize      GetScreenSize() const   { return m_sScr; }
CString*   GetFileName()           { return &m_strMapFile; }

CPoint     GetLT(const int nx, const int ny) const
{ return (GetCenter(nx, ny) - m_sTH); }
int        GetLTx(const int nx, const int ny) const
{ return (GetCx(nx, ny) - m_sTH.cx); }
int        GetLTy(const int ny) const
{ return (GetCy(ny) - m_sTH.cy); }
CPoint     GetCenter(const CPoint& point) const
{ return GetCenter(point.x, point.y); }
CPoint     GetCenter(const int nx, const int ny) const
{ return CPoint(GetCx(nx, ny), GetCy(ny)); }
int        GetCx(const int nx, const int ny) const
{ return ((ny%2 == 0) ? nx * m_sT.cx : (2*nx + 1) *
m_sTH.cx); }
int        GetCy(const int ny) const { return ny * m_sTH.cy; }
BOOL       InsideTile(const CPoint& ptTileID, const CPoint& point) const;
int        GetElevation(const CPoint& ptID) const;
CPhasedSprite* GetPS(const CPoint& ptID)
{ return m_apTile[ptID.y][ptID.x].pPS; }
int        GetEA(const CPoint& ptID) const // Elevation for Actors
{ return m_apTile[ptID.y][ptID.x].nEA; }
WORD       GetDA(const CPoint& ptID) const // Direction Attribute
{ return m_apTile[ptID.y][ptID.x].wDA; }
BOOL       CanStandOn(const CPoint& ptID) const
{ return (GetDA(ptID) != DA_CLOSED); }
BOOL       GetActorNextTileID(CPoint& ptTID, const WORD wDA) const;
CPoint     GetEntryID(LPCTSTR szFromID) const;
BOOL       IsValidTileID(const CPoint& ptTID) const
{ return ((ptTID.x >= 0) && (ptTID.x < m_sTiles.cx) &&
(ptTID.y >= 0) && (ptTID.y < m_sTiles.cy)); }

// Operations
CPoint     GetNearestTileCenter(const CPoint& pt) const;
CPoint     GetNearestTileLT(const CPoint& pt) const
{ CPoint ptCtr(GetNearestTileCenter(pt));
  return CPoint(ptCtr - m_sTH); }
CPoint     GetNearestTileIndex(const CPoint& pt) const;
CPoint     GetNearestTileIndex(CPhasedSprite* pPS) const;

void       SetTileSize(const CSize sT)
{ m_sT = sT; m_sTH.cx = m_sT.cx/2; m_sTH.cy =
m_sT.cy/2; }
BOOL       Create(const int W, const int H);
BOOL       Create(const CSize sScr) { return Create(sScr.cx,
sScr.cy); }
void       SetSpriteList(CSpriteList* pSL) { m_pSpriteList = pSL; }
}
void       SetAniList(CSpriteList* pAL) { m_pAniList = pAL; }
void       SetPalette(CPalette* pPal) { m_pPalette = pPal; }

BOOL       Load(const char* path=NULL);
BOOL       LoadRow(CTextFileBuffer& tfb, const int ny, const double
fVersion);
BOOL       Save(CStdioFile& f);

```

TileMap.h

```

void DrawElevationGraph(CDC* pDC);
void DrawActorElevationGraph(CDC* pDC);
void SetEA(const CPoint& ptID, const int nEA)
    { m_apTile[ptID.y][ptID.x].nEA = nEA; }
void IncEA(const CPoint& ptID, const int nEABY)
    { m_apTile[ptID.y][ptID.x].nEA += nEABY; }
void SynchronizeEA(const CPoint& ptID); // with sprite
elevation
    BOOL SynchronizeAllEA();
    void IncreaseElevations(const int nPixelBy);
    CPhasedSprite* HitTest(const CPoint& point);
protected:
    void InsertList(CPhasedSprite* pPS);
    void RemoveList(CPhasedSprite* pPS);

#ifdef MAPEDITOR
public:
    void RenderGrid(CDIB* pDIB, const CRect* pClipRect);
    void SetDA(const CPoint& ptID, const WORD wDA);
    BOOL RecalculatedA();
    BOOL Insert(CPhasedSprite* pPS);
//    BOOL CopyInsert(CPhasedSprite* pPS);
    void Delete(CPhasedSprite* pPS);
    BOOL MoveSpriteTo(CPhasedSprite* pPS, CPoint& ptTo);
    BOOL LoadGrid(const WORD wResId=DIB_GRID_64X32);
protected:
    void SetGridImage(const CPoint& ptID, const WORD wDA);
#endif

private:
    void DeleteTiles();
    void DrawHyperlink(CDC* pDC, const CPoint& ptC, const int nType);

    CSize m_sScr; // Screen Width, Height
    CSize m_sT; // Tile width, height
    CSize m_sTH; // Half Tile width, height

    LPTILE* m_apTile; // 2-dim array of TILES
    CSize m_sTiles; // # of tiles
    CString m_strMapFile; // filename.ext only for Version 0.99 and
below
//    BOOL m_bGrid;

    CSpriteList* m_pSpriteList;
    CSpriteList* m_pAniList;
    CPalette* m_pPalette;
};

#endif //
!defined(AFX_TILEMAP_H__D5010EC6_A1F9_11D1_80E2_080009B9F339__INCLUDED_)

```


To Compile Mall Version

Mall

====

1. Add Preprocessor definition `_MALL` to the option dialog.
Project => Settings => C/C++ => Processor definitions
2. Add some source files into the project.
DlgAni.cpp
DlgPDA.cpp
PPPaymentInfo.cpp
PPShoppingCart.cpp
3. In the files UC2View.h and UC2View.cpp,
add some `afx_msg` lines commented out as `"/* MALL */"`.
4. Change icon files in UC2/res directory.
5. Copy files in UC2/I.Mall directory to U2 directory.
6. Replace the file `u2res000.rit` with `u2res000_Mall.rit`.

```

// UC2.cpp : Defines the class behaviors for the application.
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "UC2.h"

#include "MainFrm.h"
#include "UC2Doc.h"
#include "UC2View.h"
#include "Splash.h"
#include "Stage.h"
#include "CloseDlg.h"
#include "ResMan.h"      // GetBubbleTextLimit, ...
#include <locale.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
#define IDS_ENTER_STAGE_MUSIC          133
#define IDS_ENTER_STAGE_LINK          134
#define IDD_PALETTE                    135
#define IDS_ENTER_TILE_ELEVS          135
static char THIS_FILE[] = __FILE__;
#endif

// LPCTSTR SAIT_TTS          = _T("Software\\Samsung\\TTS\\SAIT_TTS");
LPCTSTR ENT_ROOTDIR        = _T("RootDir");
// Member Info Entries
LPCTSTR ENT_NICK            = _T("Nick");
LPCTSTR ENT_REALNAME       = _T("RealName");
LPCTSTR ENT_PROFILE        = _T("Profile");
LPCTSTR ENT_HYPERLINK      = _T("Hyperlink");
LPCTSTR ENT_SEX            = _T("Sex");
LPCTSTR ENT_AGE            = _T("Age");
LPCTSTR ENT_CHARID         = _T("CharID");
LPCTSTR ENT_BUBBLEKIND     = _T("BubbleKind");
LPCTSTR ENT_CONNTYPE       = _T("ConnType");
LPCTSTR ENT_TIMEOUT        = _T("TimeOut");
LPCTSTR ENT_STAGESEC       = _T("StageSec");
LPCTSTR ENT_LASTSTAGEID    = _T("LastStageID");
LPCTSTR ENT_BUBBLELEN      = _T("BubbleLen");
LPCTSTR ENT_BUBBLETIME     = _T("BubbleTime");
//
LPCTSTR ENT_SERVER         = _T("ServerIP"); // Only for user-defined
connection type

#ifdef _MALL
const int DEFAULT_STAGE_SEC = 0;
#else
const int DEFAULT_STAGE_SEC = 10;
#endif
extern CResMan gResMan;

```

```

/////////////////////////////////////////////////////////////////
// CUC2App

BEGIN_MESSAGE_MAP(CUC2App, CWinApp)
    //{AFX_MSG_MAP(CUC2App)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    ON_COMMAND(ID_HELP_DEVELOPERS, OnHelpDevelopers)
    //}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////
// CUC2App construction

CUC2App::CUC2App()
: m_strVersion(_T("Version 1.0"))
{
    TRACE("CUC2App::CUC2App()\n");
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
    m_pStage = NULL;
    m_bPause = FALSE;
}

/////////////////////////////////////////////////////////////////
// The one and only CUC2App object

CUC2App theApp;

/////////////////////////////////////////////////////////////////
// CUC2App initialization

BOOL CUC2App::InitInstance()
{
    TRACE0("CUC2App::InitInstance()\n");
    // CG: The following block was inserted by 'Status Bar' component.
    {
        //Set up date and time defaults so they're the same as system
        defaults
        setlocale(LC_ALL, "");
    }
    // CG: The following block was added by the Splash Screen component.
    {
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);
        CSplashWnd::EnableSplashScreen(cmdInfo.m_bShowSplash);
    }

    DWORD dwVer = ::GetVersion();
    if ((dwVer & 0x800000FF) != 0x080000003)
    {

```

```

        // Not on Win32s so try to get the CreateDIBSection procedure
address.
        HMODULE hMod = ::GetModuleHandle("gdi32");
        if (!hMod || (hMod && !::GetProcAddress(hMod,
"CreateDIBSection")))
        {
            AfxMessageBox(IDS_ERROR_NOT_WIN95);
            return FALSE;
        }
    else
    {
        AfxMessageBox(IDS_ERROR_NOT_WIN95);
        return FALSE;
    }

    // First chance exception in UC2.exe (WCA_32.dll) after installing
UniWin98
    if (!AfxSocketInit())
    {
        AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
        return FALSE;
    }
    // Call AfxSocketTerm() in CUC2App::ExitInstance()

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();                // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic();          // Call this when linking to MFC
statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Samsung"));
    if (!CreateUniChatKey())
    {
        AfxMessageBox(IDS_ERROR_WRITE_REGISTRY);
        return FALSE;
    }
    GetUniChatRoot(m_strUniChatRoot);
    LoadStdProfileSettings(); // Load standard INI file options (including
MRU)

    // Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;

```

```

pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CUC2Doc),
    RUNTIME_CLASS(CMainFrame),          // main SDI frame window
    RUNTIME_CLASS(CUC2View));
AddDocTemplate(pDocTemplate);

// Parse command line for standard shell commands, DDE, file open
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);

/*
if (m_lpCmdLine[0] == '-')
{
    CString strOption(m_lpCmdLine);
    if (strOption.Find(MYPASSWORD) >= 0)        // should be >= 1
    {
        m_bLock = FALSE;
    }
}

*/

// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
    return FALSE;

// The one and only window has been initialized, so show and update it.
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

TRACE0("CUC2App::InitInstance() - Done.\n");
return TRUE;
}

// App command to run the dialog
void CUC2App::OnAppAbout()
{
    CCloseDlg dlg;
    dlg.m_strMessage.LoadString(IDS_ABOUT_TEXT);
    dlg.DoModal();
}

////////////////////////////////////
// CUC2App commands

BOOL CUC2App::OnIdle(LONG lCount)
{
    // CG: The following code inserted by 'Idle Time Processing' component.

    // Note: Do not perform lengthy tasks during OnIdle because your
    // application cannot process user input until OnIdle returns.

    // call the base class
    CWinApp::OnIdle(lCount);

    if (m_bPause)
        return FALSE;

/*
if (lCount == 0)

```

```

    {
        // TODO: add code to perform important idle time processing
    }
    else if (lCount == 100)
    {
        // TODO: add code to perform less important tasks during idle
    }
    else if (lCount == 1000)
    {
    }
}

*/

// This is the Heart of the Program!
if (m_pStage)
{
    m_pStage->TickAll();
    return TRUE;
}
// return FALSE when there is no more idle processing to do
return FALSE;
}

BOOL CUC2App::PreTranslateMessage(MSG* pMsg)
{
    // CG: The following lines were added by the Splash Screen component.
    if (CSplashWnd::PreTranslateAppMessage(pMsg))
        return TRUE;

    return CWinApp::PreTranslateMessage(pMsg);
}

/*
// returns FALSE if not found
BOOL CUC2App::GetMagicVoicePath(CString& str)
{
    // str = GetProfileString(m_strVersion, ENT_TTSDB,
    "C:\\TTS\\");
    // return TRUE;
    //

    BOOL fRet = FALSE;
    HKEY hkey;
    LONG lRet = ::RegOpenKeyEx(HKEY_LOCAL_MACHINE, SAIT_TTS, 0, KEY_READ,
    &hkey);
    if (lRet != ERROR_SUCCESS)
        return FALSE;
    DWORD dwType;
    DWORD cbBuf=1024; // This was the bug!
    LPTSTR p = str.GetBuffer(cbBuf);
    lRet = ::RegQueryValueEx(hkey, _T("DB Directory"), 0, &dwType,
    (BYTE*)p, &cbBuf);
    str = p;
    str.ReleaseBuffer(); // Surplus memory released, p is now invalid
    if (lRet == ERROR_SUCCESS)
    {
        fRet = (dwType == REG_SZ || dwType == REG_MULTI_SZ || dwType ==
        REG_EXPAND_SZ);
        ASSERT(fRet);
        int len = str.GetLength();
        if (len > 0)

```

```

        {
            if (str[len-1] != '\\')
                str += '\\';
        }
        ::RegCloseKey(hkey);
        return fRet;
    }
    */
    BOOL CUC2App::GetUniChatKey(PHKEY phkey, REGSAM samDesired)
    {
        LONG lRet = ::RegOpenKeyEx(HKEY_LOCAL_MACHINE,
            _T("Software\\Samsung\\UniChat"),
            0, samDesired, phkey);

        return (lRet == ERROR_SUCCESS);
    }

    // Check if we already have a key for UniChat or Create the key
    // Create HKEY_CURRENT_USER/Software/Samsung/UniChat Key
    // Set Values under this key
    // RootDir = "C:\\UNICHAT\\"
    BOOL CUC2App::CreateUniChatKey()
    {
        CString strRoot;
        int len = 256;
        LPTSTR pszRoot = strRoot.GetBuffer(len);
#ifdef _ROOTCURDIR // Root is current directory
        ::GetCurrentDirectory(len, pszRoot);
        strRoot.ReleaseBuffer();
        strRoot += '\\';
#else // Root is module directory
        ::GetModuleFileName(NULL, pszRoot, len);
        strRoot.ReleaseBuffer();
        int i = strRoot.ReverseFind('\\');
        if (i > 0)
        {
            pszRoot = strRoot.GetBuffer(len);
            pszRoot[i] = NULL;
            strRoot.ReleaseBuffer();
            strRoot += '\\';
        }
        else
        {
            AfxMessageBox(strRoot);
            return FALSE;
        }
#endif
        // Section, Entry, Value
        return WriteProfileString(m_strVersion, ENT_ROOTDIR, strRoot);
    }

    BOOL CUC2App::GetUniChatRoot(CString& str)
    {
        str = GetProfileString(m_strVersion, ENT_ROOTDIR);
        return TRUE;
    }

```

```

// Update registry values
BOOL CUC2App::RegSetMemberInfo(CMemberInfo& mi)
{
    if (!WriteProfileString(m_strVersion,      ENT_NICK,
        *mi.GetNick()))
        return FALSE;
    if (!WriteProfileString(m_strVersion,      ENT_REALNAME,
        *mi.GetRealName()))
        return FALSE;
    if (!WriteProfileString(m_strVersion,      ENT_PROFILE,
        *mi.GetProfile()))
        return FALSE;
    if (!WriteProfileString(m_strVersion,      ENT_HYPERLINK,
        *mi.GetHyperlink()))
        return FALSE;
    if (!WriteProfileInt(m_strVersion,         ENT_SEX,
        mi.GetSex()))
        return FALSE;
    if (!WriteProfileInt(m_strVersion,         ENT_AGE,
        mi.GetAge()))
        return FALSE;
    if (!WriteProfileInt(m_strVersion,         ENT_CHARID,
        mi.GetCharID()))
        return FALSE;
    if (!WriteProfileInt(m_strVersion,         ENT_BUBBLEKIND,
        mi.GetBubbleKind()))
        return FALSE;
    return TRUE;
}

BOOL CUC2App::RegGetMemberInfo(CMemberInfo& mi)
{
    mi.SetNick(      GetProfileString(m_strVersion,      ENT_NICK));
    mi.SetRealName(  GetProfileString(m_strVersion,
    ENT_REALNAME));
    mi.SetProfile(    GetProfileString(m_strVersion,
    ENT_PROFILE));
    mi.SetHyperlink(  GetProfileString(m_strVersion,      ENT_HYPERLINK));
    mi.SetSex(        GetProfileInt(m_strVersion,         ENT_SEX,
    0));
    mi.SetAge(        GetProfileInt(m_strVersion,         ENT_AGE,
    20));
    mi.SetCharID(     GetProfileInt(m_strVersion,         ENT_CHARID,
    0));
    mi.SetBubbleKind( GetProfileInt(m_strVersion,         ENT_BUBBLEKIND,
    0));
    return TRUE;
}

BOOL CUC2App::RegGetNick(CString& strNick)
{
    strNick = GetProfileString(m_strVersion,  ENT_NICK);
    return TRUE;
}

BOOL CUC2App::RegSetNick(const CString& strNick)
{

```



```

        return WriteProfileString(m_strVersion,    ENT_NICK,        strNick);
    }

    BOOL CUC2App::RegGetServer(CString& strServer)
    {
        strServer = GetProfileString(m_strVersion,    ENT_SERVER);
        return TRUE;
    }

    BOOL CUC2App::RegSetServer(const CString& strServer)
    {
        return WriteProfileString(m_strVersion,    ENT_SERVER,        strServer);
    }

    // Connection Type: LAN(0), MODEM(1)
    int CUC2App::RegGetConnType()
    {
        return GetProfileInt(m_strVersion,    ENT_CONNTYPE,        1);
    }

    BOOL CUC2App::RegSetConnType(const int nConnType)
    {
        return WriteProfileInt(m_strVersion,    ENT_CONNTYPE,        nConnType);
    }

    // Connection Type: Dial_Up Networking, Dedicated Line
    int CUC2App::RegGetTimeOut()
    {
        return GetProfileInt(m_strVersion,    ENT_TIMEOUT,        60);
    }

    int CUC2App::RegGetStageSec() // delay in a stage
    {
        return GetProfileInt(m_strVersion,    ENT_STAGESEC,        DEFAULT_STAGE_SEC);
    }

    BOOL CUC2App::RegGetUserID(CString& strID)
    {
        BOOL fRet = FALSE;
#ifdef _ATT
        HKEY hkey;
        LONG lRet = ::RegOpenKeyEx(HKEY_CURRENT_USER,
            _T("RemoteAccess\\Profile\\AT&T"),
                                0, KEY_READ, &hkey);

        if (lRet != ERROR_SUCCESS)
            return FALSE;
        DWORD dwType;
        DWORD cbBuf=1024;
        LPTSTR p = strID.GetBuffer(cbBuf);
        lRet = ::RegQueryValueEx(hkey, _T("User"), 0, &dwType, (BYTE*)p,
            &cbBuf);
        strID = p;
        strID.ReleaseBuffer(); // Surplus memory released, p is now invalid
        if (lRet == ERROR_SUCCESS)
        {

```

```

        fRet = (dwType == REG_SZ || dwType == REG_MULTI_SZ || dwType ==
REG_EXPAND_SZ);
        ASSERT(fRet);
        int i = strID.Find('@');
        if (i > 0)
        {
            p = strID.GetBuffer(cbBuf);
            p[i] = NULL;
            strID.ReleaseBuffer();
        }
        ::RegCloseKey(hkey);
#endif // _ATT
        return fRet;
    }

    BOOL CUC2App::RegGetLastStageID(CString& strSID)
    {
        strSID = GetProfileString(m_strVersion, ENT_LASTSTAGEID, NULL);
        return TRUE;
    }

    BOOL CUC2App::RegSetLastStageID(const CString& strSID)
    {
        return WriteProfileString(m_strVersion, ENT_LASTSTAGEID, strSID);
    }

    int CUC2App::RegGetBubbleTextLength()
    {
        return GetProfileInt(m_strVersion, ENT_BUBBLELEN,
gResMan.GetBubbleTextLimit());
    }

    BOOL CUC2App::RegSetBubbleTextLength(const int nLen)
    {
        return WriteProfileInt(m_strVersion, ENT_BUBBLELEN, nLen);
    }

    int CUC2App::RegGetBubbleTime()
    {
        return GetProfileInt(m_strVersion, ENT_BUBBLETIME,
gResMan.GetBubbleTime());
    }

    BOOL CUC2App::RegSetBubbleTime(const int nMS)
    {
        return WriteProfileInt(m_strVersion, ENT_BUBBLETIME, nMS);
    }

    int CUC2App::ExitInstance()
    {
        TRACE("CUC2App::ExitInstance()\n");
        AfxSocketTerm();
/*
        CString strModule("ChatSock.dll"); //ChatSock.dll);
        HMODULE hMod = ::GetModuleHandle(strModule);
        TRACE("GetModuleHandle(%s) returned %lx\n", strModule, hMod);

```

```
        if (hMod && ::FreeLibrary(hMod))
            TRACE("::FreeLibrary returned TRUE.\n");
    */
    return CWinApp::ExitInstance();
}

void CUC2App::WinHelp(DWORD dwData, UINT nCmd)
{
    // CCloseDlg dlg;
    // dlg.m_strMessage.LoadString(IDS_HELP_SIMPLE);
    // dlg.DoModal();
    // CWinApp::WinHelp(dwData, nCmd);    // Block
}

void CUC2App::OnHelpDevelopers()
{
    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + "Developers.htm");
    pMF->ShellBrowseURL(strFile); // "http://www.samsung.co.kr/";
}
```

UC2.h

```
// UC2.h : main header file for the UC2 application
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#if !defined(AFX_UC2_H_A1313869_A610_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_UC2_H_A1313869_A610_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CUC2App:
// See UC2.cpp for the implementation of this class
//
class CStage;
class CMemberInfo;

class CUC2App : public CWinApp
{
public:
    CUC2App();

// Attributes
    BOOL GetPause() const { return m_bPause; }

// Operations
    void SetStage(CStage* pStage)      { m_pStage = pStage; }
    void SetPause(const BOOL bPause)   { m_bPause = bPause; }

//
    BOOL GetMagicVoicePath(CString& str);
    BOOL GetUniChatKey(PHKEY phkey, REGSAM samDesired=KEY_READ);
    BOOL CreateUniChatKey();
    BOOL GetUniChatRoot(CString& str);
    BOOL RegSetMemberInfo(CMemberInfo& mi);
    BOOL RegGetMemberInfo(CMemberInfo& mi);
    BOOL RegGetNick(CString& strNick);
    BOOL RegSetNick(const CString& strNick);
    int  RegGetConnType();
    BOOL RegSetConnType(const int nConnType);
    int  RegGetTimeOut();
    int  RegGetStageSec(); // delay in a stage
    BOOL RegGetUserID(CString& strID);
    BOOL RegGetLastStageID(CString& strSID);
    BOOL RegSetLastStageID(const CString& strSID);
    BOOL RegGetServer(CString& strServer);
    BOOL RegSetServer(const CString& strServer);
};
```

```

    int             RegGetBubbleTextLength();
    BOOL    RegSetBubbleTextLength(const int nLen);
    int             RegGetBubbleTime();
    BOOL    RegSetBubbleTime(const int nMS);

    CString         m_strUniChatRoot;
    CString         m_strVersion;

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CUC2App)
public:
    virtual BOOL InitInstance();
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    virtual BOOL OnIdle(LONG lCount);
    virtual int ExitInstance();
    virtual void WinHelp(DWORD dwData, UINT nCmd = HELP_CONTEXT);
//}}AFX_VIRTUAL

// Implementation
protected:
    CStage*          m_pStage;
    BOOL             m_bPause;

    //{{AFX_MSG(CUC2App)
    afx_msg void OnAppAbout();
    afx_msg void OnHelpDevelopers();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_UC2_H__A1313869_A610_11D1_80E2_080009B9F339__INCLUDED_)

```

```

//=====
//  CUC2Socket
//  CUC2Channel
//
//  (C) Programmed by Kim, Soomin, Mar 23, 1998
//  Information Technology Institue
//  UNICHAT INC
//
//  You should not call other thread's functions except simple ones
//  like inline. That can be the cause of the deadlock between a main
thread
//  which is just waiting for this thread to terminate. But, ...
//  Do not call a complex function through m_pDoc.
//  But you should AddRef ChatSock objects before posting them with
//  WM_COMMAND, since it's become out of scope and the stack will be
//  destroyed.
//=====
#include "stdafx.h"
#include <csface.h>
#include "UC2CS.h"
#include "resource.h"

const char* szUC2Socket      = _T("UniChat Socket");      // MessageBox
title
const char* szUC2Channel    = _T("UniChat Channel");      // MessageBox
title
//-----+
// class CUC2Socket
//-----+
CUC2Socket::CUC2Socket(const HWND hWnd)
{
    TRACE0("CUC2Socket::CUC2Socket()\n");
    ASSERT(hWnd);
    m_hFrame = hWnd;
    m_hQClient = NULL;
    m_bQueryOK = TRUE;
}

CUC2Socket::~CUC2Socket()
{
    TRACE0("CUC2Socket::~CUC2Socket()\n");
}

BOOL CUC2Socket::FOnLogin()
{
    TRACE0("CUC2Socket::FOnLogin() - Connection Successful\n");
    return TRUE;
}

BOOL CUC2Socket::FOnAddChannel(PCS_MSGCHANNEL pMsg)
{
    TRACE0("CUC2Socket::FOnAddChannel\n");
    if (!m_hFrame)
        return FALSE;
    ASSERT(pMsg->picsChannel);          // ICSChannel*
    (pMsg->picsChannel)->AddRef();      // To survive asynchronous messages
    PostMessage(CSMMSG_CMD_ADDCHANNEL, 0, (LPARAM)pMsg->picsChannel);
}

```

```

        return TRUE;
    }

// Message from the host?
BOOL CUC2Socket::FOnPrivateMsg(PCS_MSGPRIVATE pMsg)
{
    TRACE0("CUC2Socket::FOnPrivateMsg\n");
    if (!m_hFrame)
        return FALSE;
    ASSERT(pMsg->picsPrivMsg);          // ICSPrivateMsg*
    (pMsg->picsPrivMsg)->AddRef();
    PostMessage(CSMMSG_CMD_PRIVATEMSG, 0, (LPARAM)pMsg->picsPrivMsg);
    return TRUE;
}

BOOL CUC2Socket::FOnInvite(PCS_MSGINVITE pcsInvite)
{
    TRACE0("CUC2Socket::FOnInvite\n");
    if (!m_hFrame)
        return FALSE;
    // pcsInvite->picsInvite          // ICSInvitation*
    (pcsInvite->picsInvite)->AddRef();
    PostMessage(CSMMSG_CMD_INVITE, 0, (LPARAM)pcsInvite->picsInvite);
    return TRUE;
}

BOOL CUC2Socket::FOnSocketError(HRESULT hr)
{
    CHAR* szError;

    switch (hr)
    {
        case E_NOTIMPL:
            szError = "Not
implemented";    break;
        case E_UNEXPECTED:
            szError = "An internal
error occurred"; break;
        case E_OUTOFMEMORY:
            szError = "Ran out of
memory";    break;
        case E_INVALIDARG:
            szError = "One or more
arguments are invalid"; break;
        case E_FAIL:
            szError = "Unspecified
error";    break;
        case S_FALSE:
            szError = "wrong
version of chatsock"; break;
        // Generic errors
        case CS_E_VERSION:
            szError = "wrong
version of chatsock"; break;
        case CS_E_WAIT:
            szError = "wait error";
            break;
        case CS_E_EVENT:
            szError = "event error";
            break;
        case CS_E_SZTOOLONG:
            szError = "string too long";
            break;
        case CS_E_EXITING:
            szError = "dll in
process of exiting"; break;
        case CS_E_NOTANSI:
            szError = "ANSI used
instead of Unicode!"; break;
    }
}

```

```

        case CS_E_NOTUNICODE:
            instead of Ansi!"; break;
        case CS_E_TOOMUCHDATA:
            max buffer bounds"; break;
        case CS_E_ILLEGAL_CHARS:
            string"; break;
        case CS_E_FIRSTCHAR:
            in IRC"; break;
        // List management errors
        case CS_E_ALREADYINLIST:
            is already in list"; break;
        case CS_E_NOTINLIST:
            found"; break;
        case CS_E_QUEUEEMPTY:
            wait q."; break;
        // Connection errors
        case CS_E_NOTCONNECTED:
            attempted on an off-line socket"; break;
        case CS_E_WINSOCKDLL:
            WinSock.dll"; break;
        case CS_E_HOSTNOTFOUND:
            by DNS"; break;
        case CS_E_SOCKETCREATE:
            break;
        case CS_E_CANTCONNECT:
            connect..server down?"; break;
        case CS_E_CANTSEND:
            sending buffer.. usually a WinSock problem"; break;
        case CS_E_TIMEOUT:
            timeout"; break;
        case CS_E_NODATA:
            error"; break;
        case CS_E_SOCKETERROR:
            error"; break;
        case CS_E_INVALIDSOCKET:
            attempted on bad ICSSocket"; break;
        case CS_E_LOSTCONNECTION:
            break;
        case CS_E_SOCKETCLOSED:
            the socket"; break;
        case CS_E_NETWORKDOWN:
            break;
        case CS_E_HOSTDROPPEDCONNECTION:
            your connection"; break;
        // catch above error!
        // Login errors
        case CS_E_NOTLOGGEDIN:
            attempted without logging into server"; break;
        case CS_E_UNKNOWNUSER:
            know you"; break;
        case CS_E_ALIASINUSE:
            already using this alias"; break;
        case CS_E_ILLEGALUSER:
            like authenticating you"; break;
        // Channel Errors
        case CS_E_CHANNELCANCEL:
            break;

szError = "Unicode used
szError = "data size exceeds
szError = "illegal chars in
szError = "first char error
szError = "inserted element
szError = "element not
szError = "no msgs in
szError = "online operation
szError = "could not load
szError = "server not found
szError = "WinSock failure";
szError = "can't
szError = "trouble
szError = "generic
szError = "NODATA"; break;
szError = "generic WinSock
szError = "operation
szError = "connection lost";
szError = "somebody closed
szError = "NETWORKDOWN";
szError = "the host server dropped
szError = "chat operation
szError = "server does not
szError = "somebody else is
szError = "server does not
szError = "CHANNELCANCEL";

```



```

        case CS_E_CREATEFAIL:
failed"; break;
        case CS_E_JOINFAIL:
failed"; break;
        case CS_E_CANCELFAIL:
break;
        case CS_E_CHANNELEXISTS:
exists"; break;
        case CS_E_CHANNELNOTFOUND:
bogus channel"; break;
        case CS_E_CANTMAKEUNIQUECHANNEL:
Channel"; break;
        case CS_E_CHANNELFULL:
break;
        case CS_E_ALREADYONCHANNEL:
channel you are already on"; break;
        case CS_E_CLOSE:
channel"; break;
        case CS_E_NOTINCHANNEL:
attempted on a channel you are not on"; break;
        case CS_E_TOOMANYCHANNELS:
on # of channels a user can be in at a time"; break;
        case CS_E_INVITEONLYCHANNEL:
this channel"; break;
        case CS_E_CHANNELBADPASS:
the channel"; break; // "bad password on the channel"
        case CS_E_NOMATCHES:
break; // On return for query
        default:
cannot be completed."; break;
    }

    SetQueryOK();
    if (m_hQClient) // If we have a query client
    {
        ::SendMessage(m_hQClient, (hr == CS_E_NOMATCHES) ?
CMD_QUERY_NOMATCHES : CMD_QUERY_ERROR, 0, (LPARAM)szError);
        m_hQClient = NULL;
    }
    else
    {
        CString str;
        str.Format("ChatSock ERROR %lx: %s\n", hr, szError);
        if (hr == CS_E_CHANNELFULL)
        {
            PostMessage(CMD_CHANNELFULL_RETRY, 0, 0);
        }
        else
        {
            ::MessageBox(NULL, str, szUC2Socket, MB_OK |
MB_ICONEXCLAMATION | MB_SYSTEMMODAL);
        }
    }
    return TRUE;
}

void CUC2Socket::SetLastQueryType(const int nType)

```

```

{
    m_LastQType = nType;
    SetQueryOK(FALSE);
    m_iItem = 0;
}

////////////////////////////////////
/////
// Initiates queries

BOOL CUC2Socket::FQueryListAllUsers(char* szFind)
{
    ASSERT(m_pics);

    if (!IsQueryOK())
    {
        CString strMsg;
        strMsg.LoadString(IDS_QUERY_LINE_BUSY);
        ::MessageBox(NULL, strMsg, szUC2Socket, MB_OK |
MB_ICONEXCLAMATION);
        return FALSE;
    }
    HRESULT hr = m_pics->HrListAllUsersA(szFind, CSROP_QUERY_CONTAINS);
    if (FAILED(hr))
    {
        FOnSocketError(hr);    // virtual function call
        return FALSE;
    }
    SetLastQueryType(Q_LIST_USERS);
    return TRUE;
}

// Lists all channels that meet the following criteria:
// a) the channel name must contain szChannel, and
// b) the channel must have a number of members within the specified
range.
BOOL CUC2Socket::FQueryListChannels(char* szChannel, DWORD dwcMin, DWORD
dwcMax)
{
    ASSERT(m_pics);

    if (!IsQueryOK())
    {
        CString strMsg;
        strMsg.LoadString(IDS_QUERY_LINE_BUSY);
        ::MessageBox(NULL, strMsg, szUC2Socket, MB_OK |
MB_ICONEXCLAMATION);
        return FALSE;
    }
    HRESULT hr = m_pics->HrListAllChannelsA(dwcMin, dwcMax, szChannel,
CSROP_QUERY_CONTAINS);
    if (FAILED(hr))
    {
        FOnSocketError(hr);    // virtual function call
        return FALSE;
    }
}

```

```

        SetLastQueryType(Q_LIST_CHANNELS);
        return TRUE;
    }

    // Lists all members in the specified channel.
    BOOL CUC2Socket::FQueryMembersInChannel(const DWORD dwID)
    {
        ASSERT(m_pics);

        if (!IsQueryOK())
        {
            CString strMsg;
            strMsg.LoadString(IDS_QUERY_LINE_BUSY);
            ::MessageBox(NULL, strMsg, szUC2Socket, MB_OK |
MB_ICONEXCLAMATION);
            return FALSE;
        }
        HRESULT hr = m_pics->HrListAllMembersA(dwID);
        if (FAILED(hr))
        {
            FOnSocketError(hr); // virtual function call
            return FALSE;
        }
        SetLastQueryType(Q_CHANNEL_MEMBERS);
        return TRUE;
    }

    BOOL CUC2Socket::FQueryMembersInChannelName(char* szChannel)
    {
        ASSERT(m_pics);

        if (!IsQueryOK())
        {
            CString strMsg;
            strMsg.LoadString(IDS_QUERY_LINE_BUSY);
            ::MessageBox(NULL, strMsg, szUC2Socket, MB_OK |
MB_ICONEXCLAMATION);
            return FALSE;
        }
        HRESULT hr = m_pics->HrListAllMembersFromNameA(szChannel);
        if (FAILED(hr))
        {
            FOnSocketError(hr); // virtual function call
            return FALSE;
        }
        SetLastQueryType(Q_CHANNEL_MEMBERS);
        return TRUE;
    }

    BOOL CUC2Socket::FQueryRealName(char* szNick)
    {
        ASSERT(m_pics);

        // if (!IsQueryOK())
        // {
        //     CString strMsg;
        //     strMsg.LoadString(IDS_QUERY_LINE_BUSY);

```

```

//      ::MessageBox(NULL, strMsg, szUC2Socket, MB_OK |
MB_ICONEXCLAMATION);
//      return FALSE;
//  }
HRESULT hr = m_pics->HrGetRealNameA(szNick);
if (FAILED(hr))
{
    FOnSocketError(hr);    // virtual function call
    return FALSE;
}
SetLastQueryType(Q_GET_REALNAME);
return TRUE;
}

////////////////////////////////////
////////

BOOL CUC2Socket::FParseQueryData(PCS_PROPERTY pcsProp)
{
    BOOL fRet = TRUE;
    ASSERT(pcsProp);
    if (!pcsProp->picsProperty)
    {
        // Null Property pointers indicate that
        // this is the last record in a data set returned by a query
        return TRUE;
    }

    // TRACE0(":::\n"); // next line

    switch (GetLastQueryType())
    {
    case Q_LIST_CHANNELS:    // CPPChannel inquires
        {
            if (!m_hQClient)
            {
                TRACE0("CPPChannel is gone\n");
                return FALSE;    // CPPChannel dialog may be
destructured...
            }
            if (FAILED(pcsProp->picsProperty-
>HrSetPrivateData((PVOID)m_iItem)))
            {
                TRACE0("HrSetPrivateData\n");
                return FALSE;
            }
        }
        TRACE("SEND=>0x%lx\n", (LPARAM)pcsProp->picsProperty);
        // ::SendMessage - The function calls the window procedure
for the specified window
        // and does not return until the window procedure has
processed the message.
        ::SendMessage(m_hQClient, CMD_QUERY_CHANNELS, 0,
(LPARAM)pcsProp->picsProperty);
        m_iItem++;
        if (pcsProp->fLastRecord)
        {

```

```

        TRACE("\nQ_LIST_CHANNELS: Got last data. (%d)\n",
m_iItem);
        SetQueryOK();
        ::SendMessage(m_hQClient, CMD_QUERY_CHANNELS_END, 0,
(LPARAM)0);
        m_hQClient = NULL;    // clear the query client
    }
    break;
case Q_LIST_USERS:    //
case Q_CHANNEL_MEMBERS: // CMemberDlg inquires
{
    if (!m_hQClient)
    {
        TRACE0("CDlgMemberList is gone\n");
        return FALSE;
    }
    if (FAILED(pcsProp->picsProperty-
>HrSetPrivateData((PVOID)m_iItem)))
    {
        TRACE0("HrSetPrivateData\n");
        return FALSE;
    }
    ::SendMessage(m_hQClient, CMD_QUERY_MEMBERS, 0,
(LPARAM)pcsProp->picsProperty);
    m_iItem = 0;
    if (pcsProp->fLastRecord)
    {
        TRACE("\nQ_LIST_USERS or Q_CHANNEL_MEMBERS: Got last
data. (%d)\n", m_iItem);
        SetQueryOK();
        HWND hDlg = m_hQClient;
        m_hQClient = NULL;    // clear the client in the
Doc
        ::SendMessage(hDlg, CMD_QUERY_MEMBERS_END, 0,
(LPARAM)0);
    }
    break;
case Q_GET_REALNAME:    // CSINDEX_PROP_REALNAME_NAME
{
    SetQueryOK();
    if (!m_hQClient)
    {
        TRACE0("CDlgMemberList is gone\n");
        return FALSE;
    }
    CS_PROPDATA cspd;
    if (FAILED(pcsProp->picsProperty->HrGetProperty(&cspd,
CSINDEX_PROP_REALNAME_REALNAME)))
        return FALSE;
    CHAR* szName = (CHAR*)cspd.pbData;
    TRACE("Q_GET_REALNAME => (%s)\n", szName);
    ::SendMessage(m_hQClient, CMD_QUERY_GET_REALNAME, 0,
(LPARAM)szName);
}
break;

```

```

        default:
            break;
        }
        return fRet;
    }

    BOOL CUC2Socket::FUnknownMessage(PCS_MSGBASE pMsg)
    {
        TRACE("CUC2Socket::FUnknownMessage\n");
        return TRUE;
    }

    //-----+
    // class CUC2Channel
    //-----+
    CUC2Channel::CUC2Channel(const HWND hWnd)
    {
        TRACE0("CUC2Channel::CUC2Channel()\n");
        ASSERT(hWnd);
        m_hFrame = hWnd;
    }

    CUC2Channel::~CUC2Channel()
    {
        TRACE0("CUC2Channel::~CUC2Channel()\n");
    }

    BOOL CUC2Channel::FOnChannelError(HRESULT hr)
    {
        // Error returned asynchronously
        CString str;
        str.Format("Channel Error %lx.\n", hr);
        TRACE(str);
        // ::MessageBox(NULL, str, szUC2Channel, MB_OK | MB_ICONEXCLAMATION);
        return TRUE;
    }

    BOOL CUC2Channel::FOnAddMember(PCS_MSGMEMBER pMsg)
    {
        TRACE0("CUC2Channel::FOnAddMember\n");
        if (!m_hFrame)
            return FALSE;
        PICS_MEMBER pM = pMsg->picsMember;
        ASSERT(pM);
        TRACE("[%s]\n", SzMemName(pM));
        pM->AddRef();
        PostMessage(CSMMSG_CMD_ADDMEMBER, 0, (LPARAM)pM);
        return TRUE;
    }

    BOOL CUC2Channel::FOnDelMember(PCS_MSGMEMBER pMsg)
    {
        TRACE0("CUC2Channel::FOnDelMember\n");
        if (!m_hFrame)
            return FALSE;
        // PICS_MEMBER pM = pMsg->picsMember;
        // ASSERT(pM);

```

```

//      pM->AddRef();
//      SendMessage(CSMSG_CMD_DELMEMBER, 0, (LPARAM)pMsg);
//      return TRUE;
}

BOOL CUC2Channel::FOnDelChannel(PCS_MSGCHANNEL pMsg)
{
    TRACE0("CUC2Channel::FOnDelChannel\n");
    if (!m_hFrame)
        return FALSE;
    PICS_CHANNEL pC = pMsg->picsChannel;
    ASSERT(pC);
    pC->AddRef();
    PostMessage(CSMSG_CMD_DELCHANNEL, 0, (LPARAM)pC);
    return TRUE;
}

BOOL CUC2Channel::FOnMemberModeChange(PCS_MSGMEMBER pMsg)
{
    TRACE0("CUC2Channel::FOnMemberModeChange\n");
    if (!m_hFrame)
        return FALSE;
    SendMessage(CSMSG_CMD_MODEMEMBER, 0, (LPARAM)pMsg);
    return TRUE;
}

BOOL CUC2Channel::FOnChannelModeChange()
{
    TRACE0("CUC2Channel::FOnChannelModeChange\n");
    if (!m_hFrame)
        return FALSE;
    SendMessage(CSMSG_CMD_MODECHANNEL, 0, 0);
    return TRUE;
}

BOOL CUC2Channel::FOnNewTopic()
{
    TRACE0("CUC2Channel::FOnNewTopic()\n");
    if (!m_hFrame)
        return FALSE;
    CString str("The topic is now: ");
    str += SzTopic();
    str += "\n";
    TRACE(str);
    //      SendMessage(CSMSG_CMD_NEWNICK, 0, (LPARAM)pMsg);
    //      return TRUE;
}

BOOL CUC2Channel::FOnAnsiTextMsg(PCS_MSG pMsg)
{
    TRACE0("CUC2Channel::FOnAnsiTextMsg\n");
    if (!m_hFrame)
        return FALSE;
    //      ASSERT(pMsg->picsFrom);
    //      SendMessage(CSMSG_CMD_TEXT_A, 0, (LPARAM)pMsg);
    //      return FALSE;
}

```

```

BOOL CUC2Channel::FOnDataMsg(PCS_MSG pMsg)
{
    TRACE0("CUC2Channel::FOnDataMsg\n");
    if (!m_hFrame)
        return FALSE;
    ASSERT(pMsg->picsFrom);
    SendMessage(CSMMSG_CMD_DATA, 0, (LPARAM)pMsg);
    return TRUE;
}

BOOL CUC2Channel::FOnAnsiWhisperTextMsg(PCS_MSGWHISPER pMsg)
{
    TRACE0("CUC2Channel::FOnAnsiWhisperTextMsg\n");
    if (!m_hFrame)
        return FALSE;
    SendMessage(CSMMSG_CMD_WHISPERTEXT_A, 0, (LPARAM)pMsg);
    return TRUE;
}

BOOL CUC2Channel::FOnAnsiWhisperDataMsg(PCS_MSGWHISPER pMsg)
{
    TRACE0("CUC2Channel::FOnAnsiWhisperDataMsg\n");
    if (!m_hFrame)
        return FALSE;
    SendMessage(CSMMSG_CMD_WHISPERDATA, 0, (LPARAM)pMsg);
    return TRUE;
}

BOOL CUC2Channel::FOnNewNick(PCS_NEWNICK pMsg)
{
    TRACE0("CUC2Channel::FOnNewNick\n");
    if (!m_hFrame)
        return FALSE;
    SendMessage(CSMMSG_CMD_NEWNICK, 0, (LPARAM)pMsg);
    return TRUE;
}

BOOL CUC2Channel::FUnknownMessage(PCS_MSGBASE pMsg)
{
    TRACE0("CUC2Channel::UnknownMessage\n");
    if (!m_hFrame)
        return FALSE;
    return TRUE;
}

```



```

//=====
//  CUC2Socket
//  CUC2Channel
//
//  (C) Programmed by Kim, , Mar 23, 1998
//  Information Technology Institue
//  UNICHAT INC
//=====
#ifndef __UC2CS_H
#define __UC2CS_H

#include "BaseSock.h"
#include "BaseChan.h"

//-----+
// class CUC2Socket

class CUC2Socket : public CBaseSocket
{
public:
    CUC2Socket(const HWND hWnd);
    virtual ~CUC2Socket();

    void                SetReceiver(const HWND hWnd) { m_hFrame = hWnd; }
    void                SetQueryClient(const HWND hWnd) { m_hQClient =
hWnd; }

    enum
    {
        Q_LIST_USERS,
        Q_LIST_CHANNELS,
        Q_CHANNEL_MEMBERS,
        Q_GET_REALNAME
    } QUERY_TYPE;

    BOOL                IsQueryOK() const { return
m_bQueryOK; }
    void                SetQueryOK(const BOOL bQOK=TRUE) { m_bQueryOK =
bQOK; }
    int                GetLastQueryType() const {
return m_LastQType; }
    void                SetLastQueryType(const int nType);

    BOOL                FQueryMembersInChannel(const DWORD dwID);
    BOOL                FQueryMembersInChannelName(char* szChannel=NULL);
    BOOL                FQueryListChannels(char* szChannel=NULL, DWORD
dwcMin=1, DWORD dwcMax=1000);
    BOOL                FQueryListAllUsers(char* szFind=NULL);
    BOOL                FQueryRealName(char* szNick);

    // Overrides
    virtual BOOL        FOnLogin();
    virtual BOOL        FOnSocketError(HRESULT hr);
    virtual BOOL        FOnAddChannel(PCS_MSGCHANNEL pMsg);
    virtual BOOL        FOnPrivateMsg(PCS_MSGPRIVATE pMsg);
    virtual BOOL        FOnInvite(PCS_MSGINVITE pcsInvite);
    virtual BOOL        FParseQueryData(PCS_PROPERTY pcsProp);
    virtual BOOL        FUnknownMessage(PCS_MSGBASE pMsg);

```

```

        BOOL                PostMessage(UINT message, WPARAM wParam, LPARAM
lParam) const                { return ::PostMessage(m_hFrame, message,
wParam, lParam); }
        BOOL                SendMessage(UINT message, WPARAM wParam, LPARAM
lParam) const                { return ::SendMessage(m_hFrame, message,
wParam, lParam); }

protected:
        HWND                m_hFrame;                // Frame Window to receive messages
        HWND                m_hQClient;                // Query Client to receive query
results
        BOOL                m_bQueryOK;                // Limit query if we didn't get the
answer yet...
        int                m_LastQType;                // What was the last query
type?
        int                m_iItem;                // current item
};

//-----+
// class CUC2Channel
class CUC2Channel : public CBaseChannel
{
public:
        CUC2Channel(const HWND hWnd);
        virtual ~CUC2Channel();

        void                SetReceiver(HWND hWnd)                { m_hFrame = hWnd; }
// Overrides
        virtual BOOL        FonChannelError(HRESULT hr);
        virtual BOOL        FonAddMember(PCS_MSGMEMBER pMsg);
        virtual BOOL        FonDelMember(PCS_MSGMEMBER pMsg);
        virtual BOOL        FonDelChannel(PCS_MSGCHANNEL pMsg);
        virtual BOOL        FonMemberModeChange(PCS_MSGMEMBER pMsg);
        virtual BOOL        FonChannelModeChange();
        virtual BOOL        FonNewTopic();
        virtual BOOL        FonAnsiTextMsg(PCS_MSG pMsg);
        virtual BOOL        FonDataMsg(PCS_MSG pMsg);
        virtual BOOL        FonAnsiWhisperTextMsg(PCS_MSGWHISPER pMsg);
        virtual BOOL        FonAnsiWhisperDataMsg(PCS_MSGWHISPER pMsg);
        virtual BOOL        FonNewNick(PCS_NEWNICK pMsg);
        virtual BOOL        FUnknownMessage(PCS_MSGBASE pMsg);

        BOOL                PostMessage(UINT message, WPARAM wParam, LPARAM
lParam) const                { return ::PostMessage(m_hFrame, message,
wParam, lParam); }
        BOOL                SendMessage(UINT message, WPARAM wParam, LPARAM
lParam) const                { return ::SendMessage(m_hFrame, message,
wParam, lParam); }

protected:
        HWND                m_hFrame;                // Frame Window to receive messages
};

```

UC2cs.h

#endif // __UC2CS_H

UC2Doc.cpp

```
// UC2Doc.cpp : implementation of the CUC2Doc class
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC.
//=====

#include "stdafx.h"
#include <afxmt.h>

#include "UC2.h"
#include "WebView.h"

#include "UC2Doc.h"
#include "UC2View.h"
#include "MainFrm.h"
#include "Parser.h"
#include "ResMan.h"
#include "Stage.h"
#include "TileMap.h"
#include "LoginDlg.h"
#include "Actor.h"
#include "EditSend.h"
#include "UC2Panel.h"
#include "UC2History.h"
#include "PSJoinChannel.h"
#include "MemberListDlg.h"
#include "ProgressDlg.h"
#include "ClosedDlg.h"
#ifdef _MALL
#include "DlgPDA.h"
#include "DlgAni.h"
#endif
#include <mmsystem.h>

#include "httpUtility.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CParser      gParser(1024*4); // to be used as an extern variable in other
files
CResMan      gResMan;

int          GetClientVersion()
{
    return CLIENT_VERSION;
}

const int MAX_SEND_CHAR=256;
const CSize DEFAULT_TILE_SIZE(64, 32);
const CSize DEFAULT_VIEW_SIZE(630, 368); // (640, 384);
```

```

#ifdef _MALL
LPCTSTR      UNICHAT_TITLE(_T("WoW - QuarterView ")); // M@ll
#else
LPCTSTR      UNICHAT_TITLE(_T("UniChat 2 "));
#endif
//LPCTSTR     USSC_NAME(_T("[u2/ussc]"));
//char*       USSC_PASSWORD = _T("qweqwe");

////////////////////////////////////
// CUC2Doc

IMPLEMENT_DYNCREATE(CUC2Doc, CDocument)

BEGIN_MESSAGE_MAP(CUC2Doc, CDocument)
    //{{AFX_MSG_MAP(CUC2Doc)
    ON_COMMAND(ID_VIEW_PAUSE, OnViewPause)
    ON_UPDATE_COMMAND_UI(ID_VIEW_PAUSE, OnUpdateViewPause)
    ON_COMMAND(ID_VIEW_DEMO, OnViewDemo)
    ON_UPDATE_COMMAND_UI(ID_VIEW_DEMO, OnUpdateViewDemo)
    ON_COMMAND(ID_CONNECT_SYNC, OnConnectSync)
    //}}AFX_MSG_MAP
    ON_COMMAND(IDC_BTN_CREATE, OnBtnCreate)
    ON_UPDATE_COMMAND_UI(IDC_BTN_CREATE, OnUpdateBtnCreate)
    ON_COMMAND(IDC_BTN_ROOM, OnBtnRoom)
    ON_UPDATE_COMMAND_UI(IDC_BTN_ROOM, OnUpdateBtnRoom)
    ON_COMMAND(IDC_BTN_MEMBER, OnBtnMember)
    ON_UPDATE_COMMAND_UI(IDC_BTN_MEMBER, OnUpdateBtnMember)
    ON_COMMAND(IDC_BTN_SOUND, OnBtnSound)
    ON_UPDATE_COMMAND_UI(IDC_BTN_SOUND, OnUpdateBtnSound)
    ON_COMMAND(IDC_BTN_HISTORY, OnBtnHistory)
    ON_UPDATE_COMMAND_UI(IDC_BTN_HISTORY, OnUpdateBtnHistory)
    ON_COMMAND(IDC_BTN_QUIT, OnBtnQuit)
    ON_UPDATE_COMMAND_UI(IDC_BTN_QUIT, OnUpdateBtnQuit)
END_MESSAGE_MAP()

////////////////////////////////////
// CUC2Doc construction/destruction

CUC2Doc::CUC2Doc()
{
    TRACE0("CUC2Doc::CUC2Doc()\n");
    m_pStage      = NULL;
    m_bSound      = TRUE;
    m_bHPanel     = FALSE;
    m_wUC2Mode    = UC2MODE_OFFLINE;
    m_pSocket     = NULL;
    m_pChannel    = NULL;
    m_wSavedState = AS_STAND | DA_BR;

    //CHANGES MADE FOR UNICHAT_2
    m_bIsJoinChannelVisible = FALSE;
}

CUC2Doc::~CUC2Doc()
{
    TRACE0("CUC2Doc::~CUC2Doc()\n");
    if (m_pStage)

```

```

    {
        EndAnimation();
        delete m_pStage;
    }
    if (m_pSocket)
    {
        TRACE0("-CUC2Doc - Destroying m_pSocket\n");
        delete m_pSocket;
    }
}

CUC2View* CUC2Doc::GetUC2View()
{
    TRACE("\nCUC2Doc::GetUC2View()");
    POSITION pos = GetFirstViewPosition();
    ASSERT(pos);
    CUC2View* pView = (CUC2View*)GetNextView(pos);
    pView = (CUC2View*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CUC2View)));
    return pView;
}

CWebView* CUC2Doc::GetWebView()
{
    TRACE("\nCUC2Doc::GetWebView()");
    POSITION pos = GetFirstViewPosition();
    ASSERT(pos);
    CWebView* pView = (CWebView*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CWebView)));
    return pView;
}

void CUC2Doc::DeleteContents()
{
    TRACE0("CUC2Doc::DeleteContents()\n");
    Disconnect();

    // if (m_pScript)
    // {
    //     TRACE0("-CUC2Doc - Destroying m_pScript\n");
    //     if (m_pScript->IsPlaying())
    //         m_pScript->Stop();
    //     delete m_pScript;
    //     m_pScript = NULL;
    // }
    #ifndef _MALL
    if (IsDemo())
    {
        AfxMessageBox(IDS_TURNING_OFF_DEMO);
        SetUC2Mode(UC2MODE_OFFLINE);
        Wait(1000);
    }
    #endif
}

```

```

        if (m_pStage)
        {
            TRACE0("~CUC2Doc - Destroying m_pStage\n");
            delete m_pStage;
            m_pStage = NULL;
        }

        SetModifiedFlag(FALSE);
        CDocument::DeleteContents();
    }

    BOOL CUC2Doc::OnNewDocument()
    {
        TRACE0("CUC2Doc::OnNewDocument()\n");
        EndAnimation();
        if (!CDocument::OnNewDocument())
            return FALSE;

        return CreateNewStage();
    }

    BOOL CUC2Doc::CreateNewStage()
    {
        TRACE("\nCreateNewStage()\n");

        CUC2View* pView = GetUC2View();
        ASSERT(pView);

        if (m_pStage)
        {
            m_pStage->DeleteStage();
            m_pStage->ClearFilename();
        }
        else
        {
            m_pStage = new CStage;
            m_pStage->Initialize(pView);
        }

        CTileMap* pTM = m_pStage->GetTileMap();
        CSize szT = pTM ? pTM->GetTileSize() : DEFAULT_TILE_SIZE;
        CSize szScr = pTM ? pTM->GetScreenSize() : DEFAULT_VIEW_SIZE;

        if (!m_pStage->CreateStage(szT, szScr))
            return FALSE;

        /*
        CString strSID;
        CUC2App* pApp = (CUC2App*)AfxGetApp();
        pApp->RegGetLastStageID(strSID);
        // Just display the last stage (not connected yet)
        // *gResMan.GetStageName(0) is the first UniChat MUD stage
        LoadStage(strSID.IsEmpty() ? *gResMan.GetStageName(0) : strSID);
        */

        return TRUE;
    }

```

```

/////////////////////////////////////////////////////////////////
// CUC2Doc serialization

void CUC2Doc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////
// CUC2Doc diagnostics

#ifdef _DEBUG
void CUC2Doc::AssertValid() const
{
    CDocument::AssertValid();
}

void CUC2Doc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

/////////////////////////////////////////////////////////////////
// CUC2Doc commands

void CUC2Doc::LoadStage(LPCTSTR szStageID)
{
    // end
    TRACE("CUC2Doc::LoadStage(%s)\n", szStageID);
    EndAnimation();
    BeginWaitCursor();

    CActor* pActor = m_pStage->GetThisActor();
    if (pActor)
        m_wSavedState = pActor->GetState(); // Save direction at previous
stage

    // Get the file for szStageID from the host

    if (m_pStage->Load(szStageID))
        // if (m_pStage->Load("000TEST"))
        {
            // <Testing...
            // m_pStage->CreateActor(0, CPoint(5, 12), TRUE, wState,
TRUE); // (nChar%4 == 0));
            // pActor = m_pStage->GetThisActor();
            // GetUC2View()->SetActor(pActor);
            // Testing>

```



```

        GetUC2View()->NewStageLoaded(); // Notify
    }
    EndWaitCursor();
    BeginAnimation();

    UpdateAllViews(NULL);
    SetTitle(*m_pStage->GetTitle());
}

void CUC2Doc::SetTitle(LPCTSTR lpszTitle)
{
    CString strTitle(UNICHAT_TITLE);
    if (lpszTitle)
        strTitle += lpszTitle;
    MakeUnichatTitle(strTitle);
    CDocument::SetTitle(strTitle);
}

void CUC2Doc::OnViewPause()
{
    PauseAnimation(!GetPause());
}

void CUC2Doc::OnUpdateViewPause(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(GetPause());
}

void CUC2Doc::EndDemo()
{
    m_bSound = FALSE;
    gResMan.SetMute(!m_bSound);
    if (!m_bSound && m_pStage)
        m_pStage->StopBGM();

    CCloseDlg dlg;
    dlg.m_strMessage.LoadString(IDS_TURNING_OFF_DEMO);
    if (dlg.DoModal() != IDOK)
        return;
    SetUC2Mode(UC2MODE_OFFLINE);
}

void CUC2Doc::PlayWaveFile(LPCTSTR szFile)
{
    CString strPath(*gResMan.GetResPath());
    CString strFile(strPath + szFile);
    ::sndPlaySound(strFile, SND_ASYNC);
}

////////// Create Channel
void CUC2Doc::OnBtnCreate()
{
    TRACE("CUC2Doc::OnBtnCreate()\n");

#ifdef _MALL
    if (IsInChannel())
    {

```

```

        CMemberListDlg dlgML;
        dlgML.SetPPChannel(this, 0, m_strStageName);
        dlgML.DoModal();
    }
    else
    {
        JoinChannel();
    }
#else
    if (IsDemo())
    {
        EndDemo();
        return;
    }
    JoinChannel();
#endif
    /*      CEditSend* pES = GetEditSend();
    if (pES)
        pES->SetFocus();
    */
}

void CUC2Doc::OnUpdateBtnCreate(CCmdUI* pCmdUI)
{
    #ifndef _MALL        // Shopping Cart
        pCmdUI->Enable((BOOL)m_pStage);
    #endif
}

////////// Rooms List
void CUC2Doc::OnBtnRoom()
{
    #ifdef _MALL        // Shopping Cart
        CString strTitle;
        strTitle.LoadString(IDS_JOIN_CHANNEL_TITLE);
        CPSJoinChannel    psjc(this, PS_JOIN_PAGE, strTitle); // List Only

        //CHANGES_MADE_FOR_UNICHAT_2
        m_bIsJoinChannelVisible = TRUE;
        psjc.DoModal();
        m_bIsJoinChannelVisible = FALSE;
    #else
        ListChannels();
    #endif
    CEditSend* pES = GetEditSend();
    if (pES)
        pES->SetFocus();

    // Testing
    //      LoadStage("preview");
}

void CUC2Doc::OnUpdateBtnRoom(CCmdUI* pCmdUI)
{
    #ifndef _MALL        // Payment Info
        pCmdUI->Enable((BOOL)m_pStage);
    #endif
}

```

```

}

////////// Members List
void CUC2Doc::OnBtnMember()
{
#ifdef _MALL // Payment Info
    CString strTitle;
    strTitle.LoadString(IDS_JOIN_CHANNEL_TITLE);
    CPSJoinChannel psjc(this, PS_CREATE_PAGE, strTitle); // List
    Only
        //CHANGES MADE FOR UNICHAT_2
        m_bIsJoinChannelVisible = TRUE;
        psjc.DoModal();
        m_bIsJoinChannelVisible = FALSE;
#else
    CMemberListDlg dlgML;
    dlgML.SetPPChannel(this, 0, m_strStageName);
    dlgML.DoModal();
#endif
    CEditSend* pES = GetEditSend();
    if (pES)
        pES->SetFocus();
}

void CUC2Doc::OnUpdateBtnMember(CCmdUI* pCmdUI)
{
#ifdef _MALL // Payment Info
    pCmdUI->Enable((BOOL)m_pStage);
#endif
}

////////// Sound On/Off
void CUC2Doc::OnBtnSound()
{
    CEditSend* pES = GetEditSend();
    if (pES)
        pES->SetFocus();
    m_bSound = !m_bSound;
    gResMan.SetMute(!m_bSound);
    if (!m_bSound && m_pStage)
        m_pStage->StopBGM();
}

void CUC2Doc::OnUpdateBtnSound(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck((BOOL)m_bSound);
}

////////// History Panel Show/Hide
void CUC2Doc::OnBtnHistory()
{
    CEditSend* pES = GetEditSend();
    if (pES)
        pES->SetFocus();
    m_bHPanel = !m_bHPanel;
    ((CMainFrame*)AfxGetMainWnd())->ShowHistoryPanel(m_bHPanel);
}

```

```

}

void CUC2Doc::OnUpdateBtnHistory(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bHPanel);
}

//////////////////// Quit
void CUC2Doc::OnBtnQuit()
{
    CEditSend* pES = GetEditSend();
    if (pES)
        pES->SetFocus();

    if (IsInChannel())
    {
        LeaveChannel(); // delete m_pChannel
        CCloseDlg dlg;
        dlg.m_strMessage.LoadString(IDS_CHANNEL_OUT);
        if (dlg.DoModal() == IDOK)
        {
            CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
            pMF->PostMessage(WM_CLOSE);
        }
    }
    else
    {
        CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
        pMF->PostMessage(WM_CLOSE);
    }
}

void CUC2Doc::OnUpdateBtnQuit(CCmdUI* pCmdUI)
{
    pCmdUI->Enable((BOOL)m_pStage);
}

////////////////////////////////////
CEditSend* CUC2Doc::GetEditSend() const
{
    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    ASSERT(pMF);
    CUC2Panel* pPanel = pMF->GetPanel();
    return (pPanel ? pPanel->GetEditSend() : NULL);
}

//CEditHistory* CUC2Doc::GetEditHistory() const
CEdit* CUC2Doc::GetEditHistory() const
{
    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
    ASSERT(pMF);
    CUC2History* pHPanel = pMF->GetHistoryPanel();
    return (pHPanel ? pHPanel->GetEditHistory() : NULL);
}

// Display message in CChatView

```

```

void CUC2Doc::DisplayText(LPCTSTR lpszText)
{
    // CEditHistory* pEdit = GetEditHistory();
    CEdit* pEdit = GetEditHistory();
    if (!pEdit)
        return;
    int count = pEdit->GetLineCount();
    /* // First Line Deletion Scheme: Flickering Problem
    if (count > MAX_CHAT_LINES)
    {
        len = pEdit->LineLength(0);
        pEdit->SetSel(0, len+2); // remove the first line
        pEdit->ReplaceSel(_T(""));
        TRACE("CChatView::Message() removed %d chars\n", len+2);
    }
    */
    if (count > 1000) // MAX_CHAT_LINES
    {
        // Save the whole text
        pEdit->SetSel(0, -1); // remove the whole lines
        pEdit->ReplaceSel(_T(""));
    }
    CString strAdd(lpszText);
    strAdd += _T("\r\n");
    int len = pEdit->GetWindowTextLength();
    pEdit->SetSel(len, len); // starting position, ending
    position
    // "tortoise:.....\n" halts the system
    pEdit->ReplaceSel(strAdd);
}

void CUC2Doc::ClearHistory()
{
    // CEditHistory* pEdit = GetEditHistory();
    CEdit* pEdit = GetEditHistory();
    if (pEdit)
    {
        pEdit->SetSel(0, -1); // select all
        pEdit->ReplaceSel(_T(""));
    }
}

// Demo
long CUC2Doc::Wait(const DWORD dwInterval)
{
    MSG msg;
    long lCount=0L;
    DWORD dwLastTick = ::GetTickCount();
    CUC2App* pApp = (CUC2App*)AfxGetApp();
    while ((m_wUC2Mode & UC2MODE_DEMO) &&
        (::GetTickCount() - dwLastTick) < dwInterval))
    {
        if (::PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
        {
            if (::GetMessage(&msg, NULL, 0, 0))
            {
                m_wUC2Mode = UC2MODE_OFFLINE;
            }
        }
    }
}

```

```

        ::PostQuitMessage(0);
        return -1;
    }
    ::TranslateMessage(&msg);
    ::DispatchMessage(&msg);
}
pApp->OnIdle(lCount++);
}
return lCount;
}

void CUC2Doc::OnUpdateViewDemo(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(IsDemo());
}

CActor* CUC2Doc::GetThisActor() const
{
    ASSERT(m_pStage);
    return m_pStage->GetThisActor();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ChatSock API
BOOL CUC2Doc::IsConnected()
{
    return (m_pSocket && m_pSocket->FConnected());
}

BOOL CUC2Doc::IsInChannel()
{
    return (m_pChannel && m_pChannel->FInChannel());
}

// Send text into opened channel
// 1. To send text chat data
// called in CEditSend::OnChar
BOOL CUC2Doc::SendText(LPCTSTR szText, const BOOL bWhisper)
{
    if (!m_pStage)
        return FALSE;
    CActor* pTA = m_pStage->GetThisActor();
    if (!pTA)
    {
        TRACE("CStage: This Actor not found!\n");
        return FALSE;
    }

    // Truncate exceeding characters
    char szTemp[MAX_SEND_CHAR+1];
    LPCSTR pSend = szText; // pointer to the text to send
    int len = lstrlen(szText);
    if (len > MAX_SEND_CHAR)
    {
        ::CopyMemory(szTemp, szText, len);
        szTemp[len] = 0;
        pSend = szTemp;
    }
}

```

```

    }

    CString strOut(m_strNick.IsEmpty() ? pSend : m_strNick + _T(":") +
pSend);
    pTA->m_mi.SetBubbleKind(pTA->m_mi.GetSex()); // Yello
    pTA->Chat(strOut); // Later, consider DBCS...
    // if (pTA->IsVoice())
    //     GetView()->ChatVoice(szText);
    DisplayText(strOut);
    return (m_pChannel ? m_pChannel->FSendAnsiText((char*)pSend) : FALSE);
}

// Send whisper message to
BOOL CUC2Doc::WhisperTo(CActor* pA, LPCTSTR szText)
{
    if (!IsInChannel())
        return FALSE;
    ASSERT(pA);
    PICS_MEMBER pM = pA->GetMember();
    if (!pM)
        return FALSE;
    CString strText(szText);
    int len = strText.GetLength();
    BOOL bRes = m_pChannel->FWhisperTo(pM, strText.GetBuffer(len));
    pM->Release();

    // Show whisper bubble only in this (sender's) client
    if (!m_pStage)
        return FALSE;
    CActor* pTA = m_pStage->GetThisActor();
    if (pTA)
    {
        pTA->m_mi.SetBubbleKind(2); // Green
        pTA->Chat(strText);
    }
    return bRes;
}

// Send State or Action Command
void CUC2Doc::SendCommand(const int nCmd)
{
    if (!m_pStage)
        return;

    CUC2View* pView = GetUC2View();

    if (nCmd == CMD_PUNCH)
    {
        if (!pView->PunchAvailable())
            return;
        pView->SaveLastHitTick();
    }

    CActor* pMe = m_pStage->GetThisActor();
    if (!pMe)
    {
        TRACE0("CUC2Doc::SendCommand - GetThisActor() failed\n");
    }
}

```

```

        return;
    }
    // If we use this scheme, ActionQueue in CActor is no more used.
    if (pMe->IsMoving())    // don't accept continuous move commands
    {
        TRACE0("->");
        return;
    }

    // CTileMap* pTM = m_pStage->GetTileMap();
    // CPoint ptTID(pTM->GetNearestTileIndex(pMe->GetEarthPoint()));
    // CPoint ptTID(pMe->m_mi.GetTileID());
    CString strCmd;
    // Get state before action!
    strCmd.Format("%c%d`(%d,%d)`%d`", UC2_SIGN_CHAR,
        nCmd, ptTID.x, ptTID.y, pMe->GetState());
    if (nCmd == CMD_PUNCH)
    {
        CActor* pFA = m_pStage->GetFrontActor();
        if (pFA)
        {
            // Show action in this machine immediately...
            pMe->Act(nCmd);
            pFA->Act(CMD_BEATEN);
            strCmd += *pFA->GetNick();
            strCmd += '`';
            SendData(strCmd);
            return;
        }
    }
    pMe->Act(nCmd);
    SendData(strCmd);
}

void CUC2Doc::SendMoveCommand(const int nCmd)
{
    if (!m_pStage)
        return;
    CActor* pMe = m_pStage->GetThisActor();
    if (!pMe)
    {
        TRACE0("CUC2Doc::SendCommand - GetThisActor() failed\n");
        return;
    }
    // If we use this scheme, ActionQueue in CActor is no more used.
    if (pMe->IsMoving())    // don't accept continuous move commands
    {
        TRACE0("->");
        return;
    }

    // CTileMap* pTM = m_pStage->GetTileMap();
    // CPoint ptTID(pTM->GetNearestTileIndex(pMe->GetEarthPoint()));
    // CPoint ptTID(pMe->m_mi.GetTileID());
    CString str;
    str.Format("%c%d`(%d,%d)`%d`", UC2_SIGN_CHAR,
        nCmd, ptTID.x, ptTID.y, pMe->GetState());
}

```



```

        CString* pStrExit = m_pStage->ActorMove(pMe, nCmd == CMD_MOVEF); //
Forward or Backward
        SendData(str);
        if (pStrExit)
        {
            JoinChannel(*pStrExit);
        }
    }

// Send data into an opened channel
// To let all the members in the channel know
// 1. To broadcast my entry in the channel I just joined
// 2. To send command data (behaviors selected)
BOOL CUC2Doc::SendData(LPCTSTR szText)
{
    if (!IsInChannel())
        return FALSE;
    DWORD len = lstrlen(szText) + 1;
    return m_pChannel->FSendData((LPBYTE)szText, len);
}

////////////////////////////////////
// ChatSock Message Handler: coming from CMainFrame
//
// Since Document cannot receive WM_COMMAND message
// Frame window will call this function for us...
// typedef IChatSocketFactory*      PICS_FACTORY;
// typedef IChatSocket*             PICS;
// typedef ICSChannel*              PICS_CHANNEL;
// typedef ICSMember*               PICS_MEMBER;
// typedef ICSQuery*                PICS_QUERY;
// typedef ICSProperty*             PICS_PROPERTY;
// typedef ICSPrivateMsg*           PICS_PRIVMSG;
// typedef ICSInvitation*           PICS_INVITATION;
//
// Typical sequence of commands from ChatSock
// 1. CSMSG_TYPE_ADDCHANNEL
// 2. CSMSG_TYPE_ADDMEMBER ...
// 3. CSMSG_TYPE_PRIVATEMSG
// 4. CSMSG_TYPE_GOTMEMLIST
// 5. CSMSG_TYPE_TEXT_A ...
// 6. CSMSG_TYPE_DELMEMBER ...
// 7. CSMSG_TYPE_DELCHANNEL

CActor*      CUC2Doc::GetActor(PICS_MEMBER pMem) const
{
    ASSERT(m_pStage);
    return m_pStage->GetActor(pMem);
}

// Load Stage on ADDCHANNEL
LRESULT CUC2Doc::OnCsAddChannel(WPARAM wParam, LPARAM lParam)
{
    PICS_CHANNEL picsChannel = (PICS_CHANNEL)lParam; // ICSChannel*
    TRACE("CUC2Doc::OnCsAddChannel - SetChannel(%lx)\n", picsChannel);
    SetChannel(picsChannel); // Don't Release pC here.
    picsChannel->Release();
}

```

```

    char* pszChName = m_pChannel->SzName();
    if (!pszChName)
        return -1;

    ClearHistory();

    CString strMsg;
    // CString strStageID(pszChName); // "[u2/0000abcd]00" or
    "[p2/0000abcd]00"
    // if (gResMan.ExtractStageID(strStageID)) // "0000abcd"
    // if (gResMan.GetStageType(pszChName) >= ST_MUD) {
    //     LoadStage(strStageID);
    //     // syc 0709
    //     CString strTemp(pszChName);
    //     int nStart = strTemp.Find('}');
    //     int length = strTemp.GetLength();
    //     char* p = strTemp.GetBuffer(length + 1);
    //     p += nStart + 1;
    //     strTemp.ReleaseBuffer();
    //     // end
    //     AfxFormatString1(strMsg, IDS_JOINED_CHANNEL, p); // pszChName); //
syc 0709
    //     GetView()->ChatVoice(strMsg);
    //     DisplayText(strMsg);
    // }
    // else {
    //     TRACE("\nCUC2Doc::OnCsAddChannel() - INVALID CHANNEL NAME");
    //     AfxFormatString1(strMsg, IDS_INVALID_CHANNEL_NAME, pszChName);
    //     AfxMessageBox(strMsg);
    // }
    // return 0;
}

// When I enter a channel, I receive member info via Private message
LRESULT CUC2Doc::OnCsPrivateMsg(WPARAM wParam, LPARAM lParam)
{
    PICS_PRIVMSG picsPrivMsg = (PICS_PRIVMSG)lParam; // ICSPrivMsg*
    ASSERT(picsPrivMsg);
    CS_PRIVMSG pm;
    if (FAILED(picsPrivMsg->HrGetMsg(&pm)))
    {
        TRACE("picsPrivMsg->HrGetMsg() failed!\n");
        picsPrivMsg->Release();
        return FALSE;
    }
    CString strData;
    if (pm.fText)
    {
        if (pm.fAnsi)
        {
            TRACE0("CSMSG_CMD_PRIVATEMSG - not ANSI\n");
            picsPrivMsg->Release();
            AfxFormatString1(strData, IDS_PRIVATE_MESSAGE,
(CHAR*)pm.pbData);
            DisplayText(strData);
            return FALSE;

```

```

    }
}
else // binary
{
    TRACE("Private MSG: binary\n");
    gParser.CopyBuffer((CHAR*)pm.pbData);
    char c;
    gParser.GetValueRightToken(c, UC2TOKEN);
    if (c == UC2_SIGN_CHAR)
    {
        char* szNickFrom;
        BOOL bAnsi;
        picsPrivMsg->HrGetMsgSender((BYTE**)&szNickFrom, &bAnsi);
        PICS_MEMBER pM=NULL;
        if (szNickFrom && m_pChannel)
        {
            PICS_CHANNEL picsChannel = m_pChannel->PChannel();
            ASSERT(picsChannel);
            picsChannel->HrGetMemberFromNameA(szNickFrom, &pM);
            picsChannel->Release();

            int nCmd;
            strData = gParser.GetValueRightToken(nCmd, UC2TOKEN);
            if (nCmd == CMD_MEMBER_INFO) // Add Actor
            {
                CActor* pA = GetActor(pM); // Find
                TRACE("\tonCsPrivateMsg - CMD_MEMBER_INFO:
                %lx\n", pM);
                if (!pA) // Not Found: New User
                {
                    CMemberInfo mi;
                    mi.SetMember(pM);
                    if (mi.SetMemberInfo(strData) &&
                    m_pStage)
                    {
                        // Create the actor who was already
                        in the channel
                        pA = m_pStage->CreateActor(mi,
                        m_pChannel->FIsMemberMe(pM));
                    }
                }
            }
            if (pM)
                pM->Release();
        }
    }
    picsPrivMsg->Release();
    return 0;
}

LRESULT CUC2Doc::OnCsQueryData(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

LRESULT CUC2Doc::OnCsInvite(WPARAM wParam, LPARAM lParam)

```

```

{
    PICS_INVITATION picsInvite = (PICS_INVITATION)lParam;
    ASSERT(picsInvite);
    BYTE* pbName;
    BYTE* pbSender;
    BOOL bAnsi;
    CHAR* szChannel;
    CHAR* szFrom;

    if (FAILED(picsInvite->HrGetChannelName(&pbName, &bAnsi)))
    {
        picsInvite->Release();
        return FALSE;
    }

    if (bAnsi)
    {
        szChannel = (CHAR*)pbName;
    }

    if (FAILED(picsInvite->HrGetSender(&pbSender, &bAnsi)))
    {
        picsInvite->Release();
        return FALSE;
    }

    if (bAnsi)
    {
        szFrom = (CHAR*)pbSender;
    }

    CString strMsg;
    AfxFormatString2(strMsg, IDS_INVITATION, szFrom, szChannel);
    // if (AfxMessageBox(strMsg, MB_YESNO) == IDYES)
    // {
    //     // Show Join Channels Dialog
    //     ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_COMMAND,
    (LPARAM) IDC_BTN_CREATE, (LPARAM) 0);
    // }
    DisplayText(strMsg);
    picsInvite->Release();
    return 0;
}

////////////////////////////////////
// Channel thread

LRESULT CUC2Doc::OnCsGotMemList(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

// We only respond to the ADDMEMBER message after GOTMEMLIST
// 1. On ADDMEMBER myself
//     SendData => Inform myself to this channel I just joined
// 2. On ADDMEMBER new member in the course of chat
//     SendPrivData => Inform myself only to this new member
LRESULT CUC2Doc::OnCsAddMember(WPARAM wParam, LPARAM lParam)

```

```

{
    ASSERT(m_pChannel);
    PICS_MEMBER pM = (PICS_MEMBER)lParam;
    ASSERT(pM);
    CString strData;
    CString str;
    CHAR* szName = m_pChannel->SzMemName(pM);

    //AddMember(picsMember);
    AfxFormatString1(str, IDS_MEMBER_ENTRY, szName);
    DisplayText(str);
    // Member info
    if (m_pChannel->FGotMemList()) // After
    {
        if (m_pChannel->FIsMemberMe(pM)) // Myself: Broadcast my info
        {
            // Set ChatID again in case this member has logged in with
            a backup id
            m_strNick = szName;
            // Load my actor
            CMemberInfo mi;
            mi.LoadMyInfo();

            PICS_CHANNEL pC = m_pChannel->PChannel();
            ASSERT(pC);
            PICS_MEMBER pM;
            pC->HrGetMe(&pM);
            pC->Release();
            ASSERT(pM);
            mi.SetMember(pM); // AddRef()
            pM->Release();
            mi.SetVersion(GetClientVersion()); // myself
            mi.SetState(m_wSavedState);

            ASSERT(m_pStage);
            CActor* pA = m_pStage->CreateActor(mi, TRUE); // myself
            CActor* pA = GetActor(pM);
            if (pA)
            {
                strData.Format("%c%d", UC2_SIGN_CHAR,
CMD_MEMBER_INFO);
                pA->m_mi.GetMemberInfo(str);
                strData += str;
                TRACE("OnCsAddMember:[SendData:Broadcast
myself]%s\n", strData);
                SendData(strData); // Broadcast in the channel
            }
            else
            {
                TRACE0("CUC2Doc::OnCsAddMember - Creation
Failure!\n");
            }
        }
        else // Inform myself to the new member
        {
            CActor* pA = GetThisActor();
            if (pA)

```

```

    {
        strData.Format("%c`%d`", UC2_SIGN_CHAR,
CMD_MEMBER_INFO);
        CMemberInfo mi;
        pA->GetMemberInfo(mi); // Get member info from this
actor
        mi.GetMemberInfo(str); // express member info in
string
        strData += str;
        DWORD dwcb = strData.GetLength() + 1;
        TRACE("OnCsAddMember: [PrivData:Inform myself]%s\n",
strData);
        m_pSocket->FSendPrivData(szName,
(BYTE*)strData.GetBuffer(dwcb-1), dwcb);
    }
    else
    {
        TRACE0("CUC2Doc::OnCsAddMember - Oops!, I'm not
created yet?\n");
    }
}
}
pM->Release();
return 0;
}

LRESULT CUC2Doc::OnCsDelMember(WPARAM wParam, LPARAM lParam)
{
    PCS_MSGMEMBER pMsg = (PCS_MSGMEMBER)lParam;
    PICS_MEMBER pM = pMsg->picsMember;
    ASSERT(pM);
    pM->AddRef();

    if (m_pChannel)
    {
        CHAR* szName = m_pChannel->SzMemName(pM);
        //DelMember(picsMember);
        CString str;
        AfxFormatString1(str, IDS_MEMBER_EXIT, szName);
        DisplayText(str);

        // Delete the member
        CActor* pA = GetActor(pM);

        if (pMsg->picsMemSrc)
        {
            PICS_MEMBER pMemKick = pMsg->picsMemSrc;
            // Someone was kicked..
            // If the member being kicked was US.. notify the user
            if (m_pChannel->FIsMemberMe(pM)) // This doesn't work
            since the member is already kicked out.
            CString strKickReason;
            CHAR* szKicker = m_pChannel->SzMemName(pMemKick);
            // Also save the reason, if any.. and only if its ANSI
            if (pMsg->pvReason && pMsg->fAnsi)
            {
                if (szKicker && pMsg->pvReason)

```

```

    {
        CString strFmt;
        strFmt.LoadString(IDS_KICKOFF_REASON);
        wsprintf(strKickReason.GetBuffer(1024),
            strFmt, szKicker, szName,

(CHAR*)pMsg->pvReason);

        strKickReason.ReleaseBuffer();
        DisplayText(strKickReason);
        if (pA == GetThisActor())
        {
            AfxMessageBox(strKickReason);
        }
    }
}

if (pA)
{
    m_pStage->DeleteActor(pA);
    //
    //
    //
    //
    m_pStage->GetRect(rc);
    GetView()->AddDirtyRegion(&rc);
    GetView()->RenderAndDrawDirtyList(); // Erase
}
pM->Release();
return 0;
}

LRESULT CUC2Doc::OnCsDelChannel(WPARAM wParam, LPARAM lParam)
{
    SetTitle(NULL);
    PICS_CHANNEL pC = (PICS_CHANNEL)lParam;
    ASSERT(pC);
    TRACE("OnCsDelChannel\n");
    pC->Release();
    return 0;
}

LRESULT CUC2Doc::OnCsModeMember(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

LRESULT CUC2Doc::OnCsModeChannel(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

// On TEXT_A - Just a text conversation
// 1. Find the actor object corresponding to the MIC member
// 2. Show the text through this actor
LRESULT CUC2Doc::OnCsTextA(WPARAM wParam, LPARAM lParam)
{
    //rao
    if(m_pChannel)
    {

```

```

PCS_MSG pMsg = (PCS_MSG)lParam;
// Obtain the sender's name
PICS_MEMBER pM = pMsg->picsFrom;
ASSERT(m_pChannel);
// if member is NULL, the message is from the channel
CHAR* szName = (pM) ? m_pChannel->SzMemName(pM) : m_pChannel-
>SzName();
ASSERT(szName);
// Is this user a Host?
CString strText;
if (pM && (pM->HrIsMemberHost() == NOERROR))
    strText = _T("*");
CString strVoice((LPCSTR)pMsg->pbData);
// if (StripComicHeader((LPCSTR)pMsg->pbData, strVoice) > 0)
// {
    strText += szName;
    strText += _T(":");
    strText += strVoice;
    DisplayText(strText);

    if (!pM)
        return 0;

    if (pM->HrIsMemberIgnored() == NOERROR)
        return 0;
    CActor* pA = GetActor(pM);
    if (pA)
    {
        if (strText.GetLength() > MAX_SEND_CHAR)
        {
            strText.GetBufferSetLength(MAX_SEND_CHAR);
            strText.ReleaseBuffer(MAX_SEND_CHAR);
        }
        pA->Chat(strText); // Later, consider DBCS...
        if (pA->IsVoice())
            GetView()->ChatVoice(strVoice, pA->GetGender());
    }
}
return 0;
}

```

```
// 1. Actors' command data
```

```
LRESULT CUC2Doc::OnCsData(WPARAM wParam, LPARAM lParam)
{

```

```

    PCS_MSG pMsg = (PCS_MSG)lParam;
    // Obtain the sender's name
    PICS_MEMBER pM = pMsg->picsFrom;

    if (pM->HrIsMemberIgnored() == NOERROR)
        return 0;
    TRACE0("Data: binary\n");
    int len = (int)pMsg->dwcbData;
    char* szBuf = new char[len+2]; // +1 for safety
    ::CopyMemory(szBuf, (LPBYTE)pMsg->pbData, len);
    szBuf[len] = NULL;

```



```

gParser.CopyBuffer(szBuf);
char c;
gParser.GetValueRightToken(c, UC2TOKEN);
if (c == UC2_SIGN_CHAR)
{
    CActor* pA = GetActor(pM);    // Sender: Find corresponding Actor
object
    int nCmd;
    CString strData(gParser.GetValueRightToken(nCmd, UC2TOKEN));
    if (nCmd == CMD_MEMBER_INFO)
    {
        TRACE("CUC2Doc::OnCsData - CMD_MEMBER_INFO: %lx\n", pM);
        if (!pA)    // Not Found: New Member
        {
            CMemberInfo mi;
            mi.SetMember(pM);
            if (mi.SetMemberInfo(strData) && m_pStage)
            {    // Create the actor who joined this channel
                CActor* pA = m_pStage->CreateActor(mi,
m_pChannel->FIsMemberMe(pM));
            }
        }
        else if (nCmd > CMD_MOVE)    // Move or Action
        {    // CMD_ACTION means
just repositioning
            TRACE("CUC2Doc::OnCsData - CMD(%d): %lx => ", nCmd, pM);
            if (pA)
            {
                WORD wState;
                CPoint ptTID;
                if (gParser.GetValueRightToken(ptTID, UC2TOKEN))
                {
                    gParser.GetValueRightToken(wState, UC2TOKEN);
                    CTileMap* pTM = m_pStage->GetTileMap();
                    CPoint ptC(pTM->GetCenter(ptTID.x, ptTID.y));
                    pA->MoveToEarth(ptC);
                    pA->SetState(wState, FALSE);    // Adjust Current
State
                }
                if (nCmd < CMD_STATE)    // MOVE
                    m_pStage->ActorMove(pA, nCmd == CMD_MOVEF);
                else
                {
                    pA->Act(nCmd);
                    if (nCmd == CMD_PUNCH)
                    {
                        CString strNickTo;
                        if (gParser.GetValueRightToken(strNickTo,
UC2TOKEN) &&
                            !strNickTo.IsEmpty())
                        {
                            CActor* pABeaten = m_pStage-
>GetActor(strNickTo);
                            if (pABeaten)
                                pABeaten->Act(CMD_BEATEN);
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    delete [] szBuf;
    return 0;
}

LRESULT CUC2Doc::OnCsWhisperText(WPARAM wParam, LPARAM lParam)
{
    PCS_MSGWHISPER pMsgW = (PCS_MSGWHISPER)lParam;
    // Obtain the sender's name
    ASSERT(pMsgW);
    // if (pMsg->dwcMem)
    PCS_MSG pMsg = (PCS_MSG)pMsgW->pcsMsg;
    ASSERT(pMsg);
    PICS_MEMBER pM = pMsg->picsFrom;
    ASSERT(m_pChannel);
    // if member is NULL, the message is from the channel
    CHAR* szName = (pM) ? m_pChannel->SzMemName(pM) : m_pChannel->SzName();
    ASSERT(szName);
    // Is this user a Host?
    CString strText;
    CString strVoice((LPCSTR)pMsg->pbData);
    if (pM->HrIsMemberHost() == NOERROR)
        strText = _T("*");
    strText += szName;
    strText += _T("=>");
    strText += strVoice;
    DisplayText(strText);

    if (!pM)
        return 0;
    if (pM->HrIsMemberIgnored() == NOERROR) // This member is ignored...
        return 0;

    CActor* pA = GetActor(pM);
    if (pA)
    {
        if (strText.GetLength() > MAX_SEND_CHAR)
        {
            strText.GetBufferSetLength(MAX_SEND_CHAR);
            strText.ReleaseBuffer(MAX_SEND_CHAR);
        }
        pA->m_mi.SetBubbleKind(2); // Green for Whisper
        pA->Chat(strText); // Later, consider DBCS...
        // if (pA->IsVoice())
        //     GetView()->ChatVoice(strVoice, pA->GetGender());
    }
    return 0;
}

LRESULT CUC2Doc::OnCsWhisperData(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

```

```

LRESULT CUC2Doc::OnCsNewTopic(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

LRESULT CUC2Doc::OnCsNewNick(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

BOOL CUC2Doc::SetChannel(PICS_CHANNEL picsChannel)
{
    // We got a new channel. Save it.
    ASSERT(m_pChannel == NULL);
    m_pChannel = new CUC2Channel(AfxGetMainWnd()->GetSafeHwnd());
    ASSERT(m_pChannel);
    // Save the ICSChannel pointer
    if (m_pChannel->FInit(picsChannel)) // PICS_CHANNEL
    {
        //CString strTitle(m_strNick);
        CString strTitle;
        strTitle += _T(":");
        strTitle += m_pChannel->SzName();

        SetTitle(strTitle);
        TRACE("You are now in channel %s\n", m_pChannel->SzName());
    }
    return TRUE;
}

////////////////////////////////////
// CUC2Doc commands

BOOL CUC2Doc::Connect()
{
    // Create a non-signaled, manual-reset event to synchronize destruction
    // m_pExitEvent = new CEvent(FALSE, TRUE);
    if (m_pSocket)
    {
        if (m_pSocket->FConnected())
        {
            AfxMessageBox("Error: Already connected.");
            return FALSE;
        }
        m_pSocket->SetReceiver(AfxGetMainWnd()->GetSafeHwnd());
    }
    else
    {
        m_pSocket = new CUC2Socket(AfxGetMainWnd()->GetSafeHwnd());
    }
    if (!m_pSocket->FInit()) // create a Socket Factory
    {
        ExitPermitted();
        return FALSE;
    }
}

```

UC2Doc.cpp

```

CUC2App* pApp = (CUC2App*)AfxGetApp();
// pApp->SetPause(TRUE);
// if (m_pScript && m_pScript->IsPlaying())
//     m_pScript->Stop();

CLoginDlg LoginDlg(m_pSocket);        // Connection (Login) Dialog
// LoginDlg.SetDoc(this);
pApp->RegGetNick(LoginDlg.m_strNickName);
if (LoginDlg.DoModal() == IDOK)
{
    // m_timeConnect = CTime::GetCurrentTime();
    // pApp->RegSetServer(ConnDlg.m_strServerIP);
    m_strNick = LoginDlg.m_strNickName;
    pApp->RegSetNick(m_strNick);
    ClearHistory();
    CString strMsg;
    AfxFormatString1(strMsg, IDS_WELCOME_TO_LOBBY, m_strNick);
    DisplayText(strMsg);

    // if (DownloadDataFiles())
    //     return FALSE;
}
else
{
    return FALSE;
}
return TRUE;
}

// Quit Connection
BOOL CUC2Doc::Disconnect()
{
    if (m_pChannel)
    {
        LeaveChannel();
    }
    if (m_pSocket)
    {
        m_pSocket->SetReceiver(NULL);
        m_pSocket->FDISCONNECT();
    }
    return TRUE;
}

BOOL CUC2Doc::ListChannels() // Just list
{
    CMemberInfo mi;
    mi.LoadMyInfo();

    CString strTitle;
    strTitle.LoadString(IDS_JOIN_CHANNEL_TITLE);
    CPSJoinChannel psjc(this, PS_JOIN_PAGE, strTitle); // List Only
    psjc.SetMemberInfo(mi);

    //CHANGES_MADE_FOR_UNICHAT_2
    m_bIsJoinChannelVisible = TRUE;
    if (psjc.DoModal() != IDOK)

```

```

    {
        m_bIsJoinChannelVisible = FALSE;
        return FALSE;
    }
    m_bIsJoinChannelVisible = FALSE;
    return TRUE;
}

void CUC2Doc::LeaveChannel()
{
    TRACE0("CUC2Doc::LeaveChannel()\n");
    m_pChannel->SetReceiver(NULL); // Don't receive message
    m_pChannel->FLeave();
    delete m_pChannel;
    m_pChannel = NULL;
}

void CUC2Doc::MakeMudChannelInfo(EC_CHANNELINFO& ecInfo)
{
    static CString strTopic;
    strTopic = m_strStageName;
    gResMan.GetStageTitle(strTopic);
    ::ZeroMemory(&ecInfo, sizeof(EC_CHANNELINFO));
    ecInfo.szName =
m_strStageName.GetBuffer(m_strStageName.GetLength() + 1);
    ecInfo.dwType = CS_CHANNEL_PUBLIC;
    ecInfo.szTopic = strTopic.GetBuffer(strTopic.GetLength() + 1);
    // UC2MUD_TOPIC;
    ecInfo.szPass = _T("");
    ecInfo.cUsersMax = MAX_MEMBERS_IN_CHANNEL; // limit # of members
in a channel
    ecInfo.dwFlags = CS_CHANNEL_FLAG_MICONLY;
}

void CUC2Doc::OnConnectSync()
{
    DownloadDataFiles();
}

// need to reload rit?
BOOL CUC2Doc::DownloadDataFiles()
{
    CProgressDlg DownloadDlg; // Download data files
    DownloadDlg.DoModal();
    if (DownloadDlg.RITModified())
    {
        if (IsDemo())
            EndDemo();
        CCloseDlg dlg;
        dlg.m_strMessage.LoadString(IDS_CLOSE_ON_NEW_RIT);
        if (dlg.DoModal() != IDOK)
            return FALSE;
        CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();
        pMF->SetAskBeforeClose(FALSE);
        pMF->PostMessage(WM_CLOSE);
        return TRUE;
    }
}

```

```

        return FALSE;
    }

    // Join a channel
    // Specifying szChannelName means automatic navigation mode (No user
    inquiry).
    BOOL CUC2Doc::JoinChannel(LPCTSTR szChannelName)
    {
        if (!IsConnected()) // ON_UPDATE_UI will prevent this
        {
            if (!Connect())
            {
                // AfxMessageBox("Connection Failed!");
                return FALSE;
            }
        }

        EC_CHANNELINFO ecInfo;
        CString strTitle;
        strTitle.LoadString(IDS_JOIN_CHANNEL_TITLE);
        CPSJoinChannel psjc(this, PS_ALL_PAGES, strTitle);
        if (szChannelName) // UniChat MUD mode
        {
            m_strStageName = szChannelName;
            gResMan.MakeStageName(m_strStageName, FALSE); // for MUD

            MakeMudChannelInfo(ecInfo);
        }
        else // User-creatable channel
        {
            CMemberInfo mi;
            mi.LoadMyInfo();

            psjc.SetMemberInfo(mi);

            //CHANGES MADE FOR UNICHAT_2
            m_bIsJoinChannelVisible = TRUE;

            // psjc.SetWizardMode();
            if (psjc.DoModal() != IDOK)
            {
                m_bIsJoinChannelVisible = FALSE;

                return FALSE;
            }
            m_bIsJoinChannelVisible = FALSE;

            psjc.GetMemberInfo(mi);
            mi.SaveMyInfo();

            if (!psjc.GetChannelInfo(ecInfo)) // Navigation mode
                return FALSE;
        }

        CString strMsg;

```

```

        HRESULT hr = ::HrVerifyCreateChannelNameA(ecInfo.szName, TRUE, FALSE);
        // fMicOnly, fLocal
        if (FAILED(hr))
        {
            AfxFormatString1(strMsg, IDS_INVALID_CHANNEL_NAME,
ecInfo.szName);
            AfxMessageBox(strMsg);
            return FALSE;
        }

        if (IsInChannel())
        {
            //          if (!szChannelName &&
            //              (AfxMessageBox(IDS_LEAVE_THIS_CHANNEL, MB_OKCANCEL) !=
IDOK))
            //              return FALSE;
            LeaveChannel();    // delete m_pChannel
        }

        CString strStageID(ecInfo.szName);    // "[u2/0000abcd]00" or
"[p2/0000abcd]"
        gResMan.ExtractStageID(strStageID);    // "0000abcd"
        // Fetch strStageID file from the Host Computer
        LoadStage(strStageID);
        if (gResMan.GetStageType(ecInfo.szName) == ST_MUD)
        {
            CUC2App* pApp = (CUC2App*)AfxGetApp();
            pApp->RegSetLastStageID(strStageID);    // Save current Stage
ID in the registry
        }

        // Set variables for OnChannelFullRetry()
        m_strStageName = ecInfo.szName;
        ASSERT(m_strStageName.GetLength() > 0);
        m_SNSuffix[0] = '0';    // Initialize
        m_SNSuffix[1] = '1';

        TRACE0("m_pSocket->FCreateJoinChannel(&ecInfo)\n");
        if (!m_pSocket->FCreateJoinChannel(&ecInfo))
        {
            AfxFormatString1(strMsg, IDS_CANNOT_JOIN_CHANNEL, ecInfo.szName);
            AfxMessageBox(strMsg);
            return FALSE;
        }

        return TRUE;
    }

LRESULT CUC2Doc::OnChannelFullRetry(WPARAM wParam, LPARAM lParam)
{
    EC_CHANNELINFO ecInfo;
    MakeMudChannelInfo(ecInfo);

    int n = lstrlen(ecInfo.szName);    // "[u2/0000abcd]00"
    if (n > 2)
    {
        ecInfo.szName[n-1] = m_SNSuffix[1];
    }
}

```

```

        ecInfo.szName[n-2] = m_SNSuffix[0];
        if (m_SNSuffix[1]++ >= '9')
        {
            m_SNSuffix[1] = '0';
            m_SNSuffix[0]++;
            if (m_SNSuffix[0] > '2')
                return -1;
        }
    }
    TRACE0("m_pSocket->FCreateJoinChannel(&ecInfo)\n");
    if (!m_pSocket->FCreateJoinChannel(&ecInfo))
    {
        CString strMsg;
        AfxFormatString1(strMsg, IDS_CANNOT_JOIN_CHANNEL, ecInfo.szName);
        AfxMessageBox(strMsg);
        return -1;
    }
    return 0;
}

void CUC2Doc::OnViewDemo()
{
    m_wUC2Mode ^= UC2MODE_DEMO;    // Toggle demo bit
    if (!IsDemo() || !m_pStage)
        return;

    if (IsDemo())
    {
        m_bSound = TRUE;
        gResMan.SetMute(!m_bSound);
        if (IsInChannel())
        {
            if (AfxMessageBox(IDS_LEAVE_THIS_CHANNEL, MB_OKCANCEL) !=
IDOK)
                return;
            LeaveChannel();    // delete m_pChannel
        }
    }

    while (IsDemo())
    {
#ifdef _MALL
#ifdef _ENGLISH
#include "DemoMallE.cpp".
#else
#include "DemoMall.cpp"
#endif
        OnBtnCreate();
#else
#include "Demo.cpp"
#endif
    }
}

#ifdef _MALL
void CUC2Doc::ShowMapDemo()
{

```



```

        CDlgAni      ani("wowmap.bmp", NULL);
        ani.SetAutoDestroy(TRUE);
        ani.SetTimeAttr(1000, 11000);
        ani.SetRelPosition(250, 50);
        ani.DoModal();
    }

void CUC2Doc::ShowIMDemo()
{
    CDlgAni      aniB("wowblist.bmp", NULL);
    aniB.SetAutoDestroy(TRUE);
    aniB.SetTimeAttr(1000, 8000);
    aniB.SetRelPosition(550, 100);
    aniB.DoModal();
    CDlgAni      ani("imwin.bmp", "imani.bmp", 9);
    ani.SetAutoDestroy(TRUE);
    ani.SetTimeAttr(500, 10000);
    ani.SetRelPosition(250, 40);
    ani.SetLT(9, 50);
    ani.DoModal();
}

void CUC2Doc::ShowTitanicDemo()
{
    PlayWaveFile("titanic.wav");
    CDlgAni      ani("liquidad.bmp", NULL);
    ani.SetAutoDestroy(TRUE);
    ani.SetTimeAttr(1000, 30000);
    ani.SetRelPosition(400, 10);
    ani.DoModal();
}
#endif

// Satya Changes Start.....
void CUC2Doc::MakeUnichatTitle(CString &Title)
{
    CString str(Title);
    int Lang, Root, Child;
    char ch;
    int Index0 = str.Find('(');
    int Index1 = str.Find(')');
    if(Index0 == -1 && Index1 == -1)
        return;
    str.Delete(Index0, Index1-Index0+1);
    Title = str;

    // modify the title according to the
    // Selected Language and Channel Categories

    Index0 = str.Find('(');
    Index1 = str.Find(')');

    if(Index0 == -1 && Index1 == -1)
        return;

    ch = str.GetAt(Index0+1);
    Lang = atoi(&ch);

```

```

ch = str.GetAt(Index0+3);
Root = atoi(&ch);
ch = str.GetAt(Index0+5);
Child = atoi(&ch);
CString Tempstr("[");

switch (Lang)
{
case 0:
    Tempstr += "En";
    break;
case 1:
    Tempstr += "Sp";
    break;
case 2:
    Tempstr += "Kr";
    break;
case 3:
    Tempstr += "Ch";
    break;
case 4:
    Tempstr += "Jp";
    break;
case 5:
    Tempstr += "Ot";
    break;
default:
    Tempstr += "En";
    break;
};
Tempstr += "]";
Tempstr += " ";
Tempstr += gResMan.GetNameChanTree(Root);
Tempstr += "->";
Tempstr += gResMan.GetNameSubItemChanTree(Root,Child);
Tempstr += " ";
str.Delete(Index0+1,5);
str.Insert(Index0+1,Tempstr);
// add Unichat Id at the last of the title
// Simply we can add at the end of the Title
str += " / ";
str += m_strNick;
Title= str;

```

```

}

```

```

void CUC2Doc::QuitChannel()
{
    //LeaveChannel();
    // Get this actor
    CActor *pthisActor = GetThisActor();

    if(pthisActor)
    {
        PICS_MEMBER pm = pthisActor->m_mi.GetMember();
    }
}

```

UC2Doc.cpp

```
    if (!pM)
        return;
    pM->HrCloseA(_T("Test"));
    pM->Release();
}
```

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

UC2Doc.h

```
// UC2Doc.h : interface of the CUC2Doc class
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#if !defined(AFX_UC2DOC_H_A131386F_A610_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_UC2DOC_H_A131386F_A610_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "UC2.h"
#include "UC2CS.h"      // EC_CHANNELINFO, CUC2Socket, CUC2Channel

class CStage;
class CUC2View;
class CEditSend;
class CEditHistory;
class CActor;
class CWebView;

enum UC2MODE
{
    UC2MODE_OFFLINE    = 0x0000,
    UC2MODE_DEMO        = 0x0001,
    UC2MODE_ONLINE      = 0x0002
};

class CUC2Doc : public CDocument
{
protected: // create from serialization only
    CUC2Doc();
    DECLARE_DYNCREATE(CUC2Doc)

// Attributes
public:
    CUC2View*          GetUC2View();
    CWebView*          GetWebView();
    CStage*             GetStage()          { return m_pStage; }
    CActor*             GetThisActor() const;

    BOOL               GetPause() const
        { return ((CUC2App*)AfxGetApp())->GetPause(); }
    BOOL               IsSoundOn() const    { return m_bSound; }
    WORD               GetUC2Mode() const   { return m_wUC2Mode; }
    BOOL               IsDemo() const       { return (m_wUC2Mode &
UC2MODE_DEMO); }
    BOOL               IsHistoryPanel() const { return m_bHPanel; }

    CEditSend*          GetEditSend() const;
    CEditHistory*        GetEditHistory() const;
    CEdit*              GetEditHistory() const;
    void                ClearHistory();

```

```

void          DisplayText(LPCTSTR lpszText);
void          SendCommand(const int nCmd);
void          SendMoveCommand(const int nCmd);
BOOL          WhisperTo(CActor* pA, LPCTSTR szText);

// ChatSock
CUC2Socket*   GetSocket()          { return m_pSocket; }
CUC2Channel*  GetChannel()         { return m_pChannel; }
BOOL          IsInChannel();
BOOL          IsConnected();

BOOL          Connect();
BOOL          Disconnect();
BOOL          ListChannels();
BOOL          JoinChannel(LPCTSTR szChannelName=NULL);
BOOL          SetChannel(PICS_CHANNEL picsChannel);

// Operations
public:
    void       BeginAnimation() const
    {
        { ((CUC2App*)AfxGetApp())->SetStage(m_pStage); }
    }

    void       EndAnimation() const
    {
        { ((CUC2App*)AfxGetApp())->SetStage(NULL); }
    }

    void       PauseAnimation(const BOOL bPause) const
    {
        { ((CUC2App*)AfxGetApp())->SetPause(bPause); }
    }

    void       SetUC2Mode(const WORD wM)      { m_wUC2Mode = wM; }
    long       Wait(const DWORD dwInterval); // for demo
    void       ToggleDemo()                  { OnViewDemo(); } // to expose
    void       EndDemo();

#ifdef _MALL
    void       ShowMapDemo();
    void       ShowIMDemo();
    void       ShowTitanicDemo();
#endif

    BOOL       SendText(LPCTSTR szText, const BOOL bWhisper=FALSE);
    BOOL       SendData(LPCTSTR szText);

    void       PlayWaveFile(LPCTSTR szFile);
    // ChatSock related methods
    CActor*    GetActor(PICS_MEMBER pMem)      const;

    // ChatSock messages
    LRESULT    OnCsAddChannel(WPARAM, LPARAM);
    LRESULT    OnCsPrivateMsg(WPARAM, LPARAM);
    LRESULT    OnCsQueryData(WPARAM, LPARAM);
    LRESULT    OnCsInvite(WPARAM, LPARAM);
    LRESULT    OnCsGotMemList(WPARAM, LPARAM);
    LRESULT    OnCsAddMember(WPARAM, LPARAM);
    LRESULT    OnCsDelMember(WPARAM, LPARAM);
    LRESULT    OnCsDelChannel(WPARAM, LPARAM);
    LRESULT    OnCsModeMember(WPARAM, LPARAM);
    LRESULT    OnCsModeChannel(WPARAM, LPARAM);
    LRESULT    OnCsTextA(WPARAM, LPARAM);
    LRESULT    OnCsData(WPARAM, LPARAM);

```

```

LRESULT OnCsWhisperText(WPARAM, LPARAM);
LRESULT OnCsWhisperData(WPARAM, LPARAM);
LRESULT OnCsNewTopic(WPARAM, LPARAM);
LRESULT OnCsNewNick(WPARAM, LPARAM);
LRESULT OnChannelFullRetry(WPARAM, LPARAM);

//changed by Rao
void LeaveChannel();
void OnBtnCreate();

//CHANGES_MADE_FOR_UNICHAT_2
BOOL m_bIsJoinChannelVisible;
protected:
    void LoadStage(LPCTSTR szStageID);

    void MakeMudChannelInfo(EC_CHANNELINFO& ecInfo);
    BOOL CreateNewStage();
    BOOL DownloadDataFiles();

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CUC2Doc)
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual void SetTitle(LPCTSTR lpszTitle);
    virtual void DeleteContents();
//}}AFX_VIRTUAL

// Implementation
public:
    void QuitChannel();
    virtual ~CUC2Doc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
private:
    CString m_RealName;
    void MakeUnichatTitle(CString& Title);
    CStage* m_pStage;
    BOOL m_bSound;
    BOOL m_bHPanel; // Is History Panel
    WORD m_wUC2Mode;
    WORD m_wSavedState;
    CString m_strNick;
    CString m_strStageName; // to join the channel
    char m_SNSuffix[2]; // StageName suffix

    // ChatSock API
    CUC2Socket* m_pSocket;
    CUC2Channel* m_pChannel;

// Generated message map functions
protected:

```

```

//{{AFX_MSG(CUC2Doc)
afx_msg void OnViewPause();
afx_msg void OnUpdateViewPause(CCmdUI* pCmdUI);
afx_msg void OnViewDemo();
afx_msg void OnUpdateViewDemo(CCmdUI* pCmdUI);
afx_msg void OnConnectSync();
//}}AFX_MSG
//void OnBtnCreate();
void OnUpdateBtnCreate(CCmdUI* pCmdUI);
void OnBtnRoom();
void OnUpdateBtnRoom(CCmdUI* pCmdUI);
void OnBtnMember();
void OnUpdateBtnMember(CCmdUI* pCmdUI);
void OnBtnSound();
void OnUpdateBtnSound(CCmdUI* pCmdUI);
void OnBtnHistory();
void OnUpdateBtnHistory(CCmdUI* pCmdUI);
void OnBtnQuit();
void OnUpdateBtnQuit(CCmdUI* pCmdUI);
DECLARE_MESSAGE_MAP()

private:
    CString m_sRoomMainCategory; // Main Room Catigorey
    CString m_sRoomSubCategory; // Sub Room Catigory
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_UC2DOC_H__A131386F_A610_11D1_80E2_080009B9F339__INCLUDED_

```

UC2History.cpp

```
//      CUC2History: History Panel for UniChat 2
//
//=====
//      (C) Programmed by Kim, Soomin, Mar 18, 1996
//      Information Technology Institue
//      UNICHAT.COM
//=====

#include "stdafx.h"
#include "UC2.h"
#include "UC2History.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

const CRect RECT_PANEL(0, 0, 180, 372);
////////////////////////////////////

CUC2History::CUC2History()
{
    m_bInitialized = FALSE;
}

CUC2History::~CUC2History()
{
}

BEGIN_MESSAGE_MAP(CUC2History, CDialogBar)
    //{AFX_MSG_MAP(CUC2History)
    ON_WM_SIZE()
    //{AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CUC2History message handlers
BOOL CUC2History::Create(CWnd* pParentWnd, UINT nStyle)
{
    TRACE0("CUC2History::Create()\n");
    BOOL bRes = CDialogBar::Create(pParentWnd, CG_IDD_HISTORYPANEL,
                                   nStyle, CG_ID_VIEW_HISTORYPANEL);
    // The DialogBar has been created.
    return bRes;
}

// Called by the parent after creation of CUC2History object and window
BOOL CUC2History::InitControls()
{
    CWnd* pW = GetDlgItem(IDC_EDIT_HISTORY);
    CRect rcEdit(4, 6, RECT_PANEL.Width()-4, RECT_PANEL.Height()-4);
    pW->SetWindowPos(NULL, rcEdit.left, rcEdit.top, rcEdit.Width(),
rcEdit.Height(),
                                SWP_NOZORDER | SWP_NOACTIVATE);
}
```


UC2History.cpp

```
// Window has been created...
// if (!m_eHistory.SubclassDlgItem(IDC_EDIT_HISTORY, this))
// {
//     TRACE0("m_EditHistory.SubclassDlgItem failed!\n");
//     return FALSE;
// }
m_bInitialized = TRUE;
return TRUE;
}

void CUC2History::OnSize(UINT nType, int cx, int cy)
{
    CDialogBar::OnSize(nType, cx, cy);
    SetWindowPos(NULL, RECT_PANEL.left, RECT_PANEL.top,
        RECT_PANEL.Width(), RECT_PANEL.Height(),
        SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
}
```

UC2History.H

```
//=====
//      (C) Programmed by Kim, Soomin, Mar 18, 1996
//      Information Technology Institue
//      UNICHAT INC
//=====

#if
!defined(AFX_UC2HISTORY_H_4B176645_BE79_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_UC2HISTORY_H_4B176645_BE79_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// UC2History.h : header file
//

// #include "EditHistory.h"

////////////////////////////////////
// CUC2History window

class CUC2History : public CDialogBar    // CControlBar - CWnd
{
// Construction
public:
    CUC2History();

// Attributes
public:
    // CEditHistory*      GetEditHistory()
    CEdit*      GetEditHistory()
    // { return m_bInitialized ? &m_eHistory : NULL; }
    { return m_bInitialized ? (CEdit*)GetDlgItem(IDC_EDIT_HISTORY) :
    NULL; }
// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUC2History)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CUC2History();

    BOOL      Create(CWnd* pParentWnd, UINT nStyle);
    BOOL      InitControls();

    // Generated message map functions
protected:
    // CEditHistory      m_eHistory;
    // CEdit      m_eHistory;
    BOOL      m_bInitialized;

    //{{AFX_MSG(CUC2History)
    afx_msg void OnSize(UINT nType, int cx, int cy);

```

UC2History.H

```
    //{AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_UC2HISTORY_H__4B176645_BE79_11D1_80E2_080009B9F339__INCLUDED_
```

UC2Messages.h

```
//
// UniChat: UniChat 2 Messages Definition
//
// (C) Programmed by Kim, Soomin, Feb 1998
// SDS Media Lab
// Information Technology Institue
// UNICHAT INC

#ifndef __UC2MESSAGES_H
#define __UC2MESSAGES_H

#include <csface.h>

////////////////////////////////////
////////////////////////////////////
// See resource.h
// where #define _APS_NEXT_COMMAND_VALUE 32798 // 0x8000
// this message should be in the range WM_USER(0x0400) to 0x7fff
#define BASE (WM_USER + 200)
const UINT CS_CMD_START = BASE;
const UINT CSMSG_CMD_ERROR = BASE + CSMSG_TYPE_ERROR;
const UINT CSMSG_CMD_LOGIN = BASE + CSMSG_TYPE_LOGIN;
const UINT CSMSG_CMD_TEXT_A = BASE + CSMSG_TYPE_TEXT_A;
const UINT CSMSG_CMD_TEXT_W = BASE + CSMSG_TYPE_TEXT_W;
const UINT CSMSG_CMD_DATA = BASE + CSMSG_TYPE_DATA;
const UINT CSMSG_CMD_ADDCHANNEL = BASE + CSMSG_TYPE_ADDCHANNEL;
const UINT CSMSG_CMD_ADDMEMBER = BASE +
CSMSG_TYPE_ADDMEMBER;
const UINT CSMSG_CMD_GOTMEMLIST = BASE + CSMSG_TYPE_GOTMEMLIST;
const UINT CSMSG_CMD_DELMEMBER = BASE +
CSMSG_TYPE_DELMEMBER;
const UINT CSMSG_CMD_DELCHANNEL = BASE + CSMSG_TYPE_DELCHANNEL;
const UINT CSMSG_CMD_MODEMEMBER = BASE + CSMSG_TYPE_MODEMEMBER;
const UINT CSMSG_CMD_MODECHANNEL = BASE + CSMSG_TYPE_MODECHANNEL;
const UINT CSMSG_CMD_WHISPERTEXT_A = BASE + CSMSG_TYPE_WHISPERTEXT_A;
const UINT CSMSG_CMD_WHISPERTEXT_W = BASE + CSMSG_TYPE_WHISPERTEXT_W;
const UINT CSMSG_CMD_WHISPERDATA = BASE + CSMSG_TYPE_WHISPERDATA;
const UINT CSMSG_CMD_NEWTOPIC = BASE + CSMSG_TYPE_NEWTOPIC;
const UINT CSMSG_CMD_PROPERTYDATA = BASE + CSMSG_TYPE_PROPERTYDATA;
const UINT CSMSG_CMD_QUERYDATA = BASE +
CSMSG_TYPE_QUERYDATA;
const UINT CSMSG_CMD_PRIVATEMSG = BASE + CSMSG_TYPE_PRIVATEMSG;
const UINT CSMSG_CMD_NEWNICK = BASE + CSMSG_TYPE_NEWNICK;
const UINT CSMSG_CMD_INVITE = BASE + CSMSG_TYPE_INVITE;
const UINT CSMSG_CMD_SERVERMSG_TEXT_A = BASE +
CSMSG_TYPE_SERVERMSG_TEXT_A;
const UINT CSMSG_CMD_SERVERMSG_TEXT_W = BASE +
CSMSG_TYPE_SERVERMSG_TEXT_W;
#undef BASE

#define BASE (CSMSG_CMD_SERVERMSG_TEXT_W + 10)
const UINT CMD_CONNECT_CONNECTING = BASE;
const UINT CMD_CONNECT_LOGIN = BASE + 1;
const UINT CMD_CONNECT_BACKUPID = BASE + 2;
const UINT CMD_CONNECT_FAILURE = BASE + 3;

const UINT CMD_QUERY_ERROR = BASE + 10;
```

UC2Messages.h

```
const UINT  CMD_QUERY_NOMATCHES          = BASE + 11;
const UINT  CMD_QUERY_CHANNELS           = BASE + 12;
const UINT  CMD_QUERY_CHANNELS_END       = BASE + 13;
const UINT  CMD_QUERY_MEMBERS            = BASE + 14;
const UINT  CMD_QUERY_MEMBERS_END        = BASE + 15;
const UINT  CMD_QUERY_GET_REALNAME       = BASE + 16;

const UINT  CMD_CHANNELFULL_RETRY        = BASE + 17;      // Retry

const UINT  CS_CMD_END                   = CMD_QUERY_GET_REALNAME;
#undef BASE
```

```
const char ARG_TOKEN    = '\0';      // token for parsing
```

```
// Parameter passed to ClientSockThread telling it which directive
// to send and to whom (either DAS or other client).
```

```
enum CLIENT_DIR
{
    REQ_INVITE=1,
    GET_HOMEFILE,
};
```

```
enum CALLSOCK_CMD // if you change this change the definitions in DAS as
well.
```

```
{
    GET_FILE=1,
    QRY_DB,
    CHK_VERSION,
    REG_ONLINE,
    REG_OFFLINE,

    AUTH_ALIAS,
    NEW_ALIAS,
    FIND_ALIAS,
    FIND_BY_FIELD,
    QRY_USERINFO,
    QRY_USERADDRESS,
    QRY_USERPROFILE,
    QRY_ISONLINE,

    UPD_MYINFO,
    UPD_USERADDRESS,
    UPD_USERPROFILE,

    REG_INFO,
};
```

```
enum DAS_RETCODE // if you change this change the definitions in DAS as
well.
```

```
{
    ALIAS_DENY = -1,
    ALIAS_UNKNOWN = 0,
```

UC2Messages.h

```

    ALIAS_OK,
    ALIAS_NEW,
    ALIAS_APPROVE,
    ALIAS_TAKEN,

    INFO_OK,
    INFO_NO,

    UPD_OK,
    UPD_NO,

    ONLINE_ACK,
    NEG_ACK,
    NO_RESP
};

enum ACTOR_COMMANDS
{
    // Command Enumerators
    CMD_BEGIN = 0,
    // Management
    CMD_MEMBER_INFO, //
X`nCmd`nVersion`nCharID`nBubbleKind`strHandle`strRealName`strProfile`ptTID`wS
tate
    CMD_MEMBER_ACTOR, // X`nCmd`nCharID`nBubbleKind (changed his Actor)
    CMD_NEWS, // X`nCmd`message
    // Position Move
    CMD_MOVE,
    CMD_MOVEF, // X`nCmd`ptTID`wState
    CMD_MOVEB,
    CMD_RES_MOVE,
    // State Change
    CMD_STATE,
    CMD_STAND, // State
    CMD_MORPH,
    CMD_DOZE,
    CMD_TURNL,
    CMD_TURNR,
    CMD_RES_STATE, // reserved
    // Actions
    CMD_ACTION, // CMD_ACTION is for Just repositioning message
    // Y`nCmd`ptTID`wState` (to verify
    synchronization)
    CMD_CHAT,
    CMD_ENTER,
    CMD_EXIT,
    CMD_SMILE,
    CMD_MAD,
    CMD_HELLO,
    CMD_CRY,
    CMD_SCRATCH,
    CMD_PICK,
    CMD_SPECIAL,
    CMD_PUNCH, // Y`nCmd`ptTID`wState`NickTo`
    CMD_BEATEN,
    CMD_RES_ACTION,
    CMD_END
};

```

```

//      BL                      [BR]
//      C
//      [FR]                    FL
enum ACTOR_STATES
{
    // for directions, use DIRECTION_ATTRIBUTES in TileMap.h
    AS_STAND      = 0x0100,
    AS_MORPH      = 0x0200,
    AS_DOZE       = 0x0400,
    AS_MASK       = 0x0F00    // States Mask
};

enum DIRECTION_ATTRIBUTES
{
    DA_FR         = 0x0001,    // Forward Right
    DA_FL         = 0x0002,
    DA_BR         = 0x0004,    // Backward
    DA_BL         = 0x0008,
    DA_OPEN       = 0x000F,    // open tile
    DA_CLOSED     = 0x0000,    // closed tile (can't go any direction)
    DA_MASK       = 0x00FF,
    DA_FORWARD    = DA_FR | DA_FL,
    DA_RIGHT      = DA_FR | DA_BR
};

const char UC2_SIGN_CHAR = 'Y';           // 'X' for UniChat 1.1
const char UC2TOKEN      = '`';          // token for parsing

// 2.00 Beta2, 2.03 Beta3, 2.11 (Aug 1, 1998)...
#ifdef _MALL
const int CLIENT_VERSION = 220;
#else
const int CLIENT_VERSION = 221;          // 211 was the last version for UNITEL
#endif

const int MAX_MEMBERS_IN_CHANNEL = 15;

#define UC2PASSWORD          _T("soomin")
#define UC2MUD_TOPIC        _T("UniChat MUD");

#endif          // __UC2MESSAGES_H

////////////////////////////////////

//
// On ADDCHANNEL
// - Broadcast (HrSendData) CMD_MEMBER_INFO in the channel
//
// or
//
// On ADDMEMBER
// if (me)
//     HrSendData (CMD_MEMBER_INFO) to this channel
// else
//     HrSendPrivMsg (CMD_MEMBER_INFO) to the member
//

```

UC2Panel.cpp

```
//
//      CUC2Panel: Control Panel for UniChat 2
//
//      (C) Programmed by Kim, Soomin, Feb 27, 1996
//      SDS Media Lab
//      Information Technology Institue
//      UNICHAT NETWORKS INC
//
// UC2Panel.cpp : implementation file
// On ToolTip support:
//      MFC42.dll 97-08-23 or MFC42d.dll (debug) 97-01-03
//      Later versions have the bug that do not display tooltips
//      for control bars in MainFrame.

#include "stdafx.h"
#include "UC2.h"
#include "UC2Panel.h"
#include "MainFrm.h"
#include "ResMan.h"

#include "UC2Ani/DIB.h"
#include "UC2Ani/DIBPal.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan gResMan;

const CRect RECT_EDIT(16, 15, 376, 38);

#ifdef _MALL
LPCTSTR BMP_PANEL_BACK = "MPanelBk.bmp";
LPCTSTR FMT_BMP_BTN_CP = "%sMBtnCP%d.bmp";
#else
LPCTSTR BMP_PANEL_BACK = "U2Panel|PanelBk.bmp";
LPCTSTR FMT_BMP_BTN_CP = "%sU2Panel|BtnCP%d.bmp";
#endif

////////////////////////////////////
// CUC2Panel

CUC2Panel::CUC2Panel()
{
    TRACE("CUC2Panel::CUC2Panel()\n");
    m_bInitialized = FALSE;
    m_pDIB = NULL;
}

CUC2Panel::~CUC2Panel()
{
    TRACE("CUC2Panel::~CUC2Panel()\n");
    if (m_pDIB)
        delete m_pDIB;
}
```



```

BEGIN_MESSAGE_MAP(CUC2Panel, CDialogBar)
    //{AFX_MSG_MAP(CUC2Panel)
    ON_WM_SETFOCUS()
    ON_WM_QUERYNEWPALETTE()
    ON_WM_PALETTECHANGED()
    ON_WM_SIZE()
    ON_WM_MOVE()
    ON_WM_ERASEBKGD()
    //}AFX_MSG_MAP
    // ON_NOTIFY(TTN_NEEDTEXT, 0, OnToolTipNotify)
END_MESSAGE_MAP()

////////////////////////////////////

BOOL CUC2Panel::Create(CWnd* pParentWnd, UINT nStyle)
{
    TRACE0("CUC2Panel::Create()\n");
    BOOL bRes = CDialogBar::Create(pParentWnd, CG_IDD_CONTROLPANEL,
                                   nStyle, CG_ID_VIEW_CONTROLPANEL);

    // The DialogBar has been created.
    return bRes;
}

CPalette* CUC2Panel::GetPalette()
{
    return ((CMainFrame*)AfxGetMainWnd())->GetPalette();
}

////////////////////////////////////
// CUC2Panel message handlers

void CUC2Panel::OnSetFocus(CWnd* pOldWnd)
{
    CDialogBar::OnSetFocus(pOldWnd);

    m_eSend.SetFocus();
}

void CUC2Panel::OnPaletteChanged(CWnd* pFocusWnd)
{
    if (!m_pDIB)
        return;
    // TRACE("CPXControlPanel::OnPaletteChanged\n");
    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CUC2Panel::OnQueryNewPalette()
{
    if (!m_pDIB)
        return FALSE;
    // TRACE("CPXControlPanel::OnQueryNewPalette\n");
    CPalette* pPal = GetPalette();
    if (pPal)
    {

```

```

        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(pPal, TRUE);
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
//        if (u)
//        {
//            // Some colors changed so we need to do a repaint.
//            Invalidate(); // Repaint the lot.
//            return TRUE; // Say we did something.
//        }
        return FALSE; // Say we did nothing.
    }

// Called by the parent after creation of CUC2Panel object and window
// Load images and initializes controls
BOOL CUC2Panel::InitControls()
{
    TRACE("CUC2Panel::InitControls()\n");
    CString strPath(*gResMan.GetResPath());
    CString strBtnBmp;
    for (int i=0; i < 6; i++)
    {
        // Two States (Normal, Selected)
        strBtnBmp.Format(FMT_BMP_BTN_CP, strPath, i);
        if (!m_aButton[i].Load(strBtnBmp, 2))
        {
            strBtnBmp += " not found!";
            AfxMessageBox(strBtnBmp);
            return FALSE;
        }
        gResMan.LoadMasterPalette(m_aButton[i].GetDIB());
        m_aButton[i].SetPalette(GetPalette());
    }
    if (m_pDIB)
        delete m_pDIB;
    strPath += BMP_PANEL_BACK;
//    TRACE("%s\n", strPath);
    m_pDIB = new CDIB;
    if (!m_pDIB->Load(strPath))
    {
        delete m_pDIB;
        m_pDIB = NULL;
        strPath + " not found!";
        AfxMessageBox(strPath);
        return FALSE;
    }
    gResMan.LoadMasterPalette(m_pDIB);
    SetWindowPos(NULL, 0, 0, m_pDIB->GetWidth(), m_pDIB->GetHeight(),
        SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);

    CWnd* pW = GetDlgItem(IDC_EDIT_SEND);
    pW->SetWindowPos(NULL, RECT_EDIT.left, RECT_EDIT.top,
        RECT_EDIT.Width(), RECT_EDIT.Height(),
        SWP_NOZORDER | SWP_NOACTIVATE);

    // Window has been created...

```

```

    if (!m_eSend.SubclassDlgItem(IDC_EDIT_SEND, this))
    {
        TRACE0("m_eSend.SubclassDlgItem failed!\n");
        return FALSE;
    }

    // Subclass 6 buttons
#ifdef _MALL
    CPoint ptLT(383, 6);
    int w = 42;
#else
    CPoint ptLT(390, 11);
    int w = 30+11;
#endif
    for (i=0; i < 6; i++)
    {
        if (!m_aButton[i].SubclassDlgItem(IDC_BTN_CREATE + i, this))
        {
            TRACE("m_aButton[%d].SubclassDlgItem failed!\n", i);
            return FALSE;
        }
        m_aButton[i].MoveResize(ptLT);
        ptLT.x += w;
    }
    // EnableToolTips();
    m_bInitialized = TRUE;
    return TRUE;
}

void CUC2Panel::OnSize(UINT nType, int cx, int cy)
{
    TRACE("CUC2Panel::OnSize(%d,%d)\n", cx, cy);
    CDialogBar::OnSize(nType, cx, cy);

    SetWindowPos(NULL, 0, 0, 636, 49-1, SWP_NOMOVE | SWP_NOZORDER |
SWP_NOACTIVATE);
}

void CUC2Panel::OnMove(int x, int y)
{
    TRACE("CUC2Panel::OnMove(%d,%d)\n", x, y);
    CDialogBar::OnMove(x, y);
}

BOOL CUC2Panel::OnEraseBkgnd(CDC* pDC)
{
    CDialogBar::OnEraseBkgnd(pDC);

    // Make sure we have what we need to do a paint.
    if (!m_pDIB)
    {
        TRACE("CUC2Panel::OnPaint() - No DIB.\n");
        return FALSE;
    }

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;

```

```

        CPalette* pPal = GetPalette();
        if (pPal)
        {
            pPalOld = pDC->SelectPalette(pPal, FALSE);        //
bForceBackground = FALSE
            pDC->RealizePalette(); // we realize in response to
WM_QUERYNEWPALETTE
        }
        m_pDIB->Draw(pDC, 0, 0);

        // Select old palette if we altered it.
        if (pPalOld)
            pDC->SelectPalette(pPalOld, FALSE);
        return TRUE;
    }

/*
int CUC2Panel::OnToolHitTest(CPoint point, TOOLINFO* pTI) const
{
    CRect rc(390, 11, 390+30, 11+30);
    if (rc.PtInRect(point))
    {
        ASSERT(pTI);
        pTI->hwnd = m_hWnd;
        pTI->rect = rc;
        pTI->uId = IDC_BTN_CREATE;
        pTI->lpszText = LPSTR_TEXTCALLBACK;
        return 1;
    }
    return CDialogBar::OnToolHitTest(point, pTI);
}

void CUC2Panel::OnToolTipNotify(UINT id, NMHDR* pNMH, LRESULT* pResult)
{
    TOOLTIPTEXT* pTTT = (TOOLTIPTEXT*)pNMH;
    UINT nID = pNMH->idFrom;
    if (pTTT->uFlags & TTF_IDISHWND)
    {
        nID = ::GetDlgCtrlID((HWND)nID);
        ASSERT(nID);
    }
    pTTT->lpszText = MAKEINTRESOURCE(nID);
    pTTT->hinst = AfxGetResourceHandle();
}
*/

```

UC2Panel.h

```
//=====
//      (C) Programmed by Kim, KYUNAM, Feb, 1998
//      Information Technology Institue
//      UNICHAT INC
//=====

#if !defined(AFX_UC2PANEL_H__CB0995A1_AF81_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_UC2PANEL_H__CB0995A1_AF81_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// UC2Panel.h : header file
//
#include "UC2Ani/PhSprite.h"
#include "UC2Ani/PSButton.h"
#include "UC2Ani/DIB.h"
#include "EditSend.h"

////////////////////////////////////
// CUC2Panel window

class CUC2Panel : public CDialogBar // CControlBar - CWnd
{
// Construction
public:
    CUC2Panel();

// Attributes
public:

// Operations
public:
    BOOL        Create(CWnd* pParentWnd, UINT nStyle);
    BOOL        InitControls();
    CEditSend*  GetEditSend()    { return m_bInitialized ? &m_eSend :
NULL; }

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CUC2Panel)
    //{AFX_VIRTUAL

// Implementation
public:
    virtual ~CUC2Panel();

protected:
    CPalette*    GetPalette();

    CDIB*        m_pDIB;
    CPSButton    m_aButton[6];    // CButton derived
    CEditSend    m_eSend;        // CEdit derived
    BOOL        m_bInitialized;
    // Generated message map functions
protected:
    //{AFX_MSG(CUC2Panel)
```

UC2Panel.h

```

    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg BOOL OnQueryNewPalette();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnMove(int x, int y);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    ///}AFX_MSG
//      int      OnToolHitTest(CPoint point, TOOLINFO* pTI) const;
//      void OnToolTipNotify(UINT id, NMHDR* pNMH, LRESULT* pResult);
//      DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_UC2PANEL_H__CB0995A1_AF81_11D1_80E2_080009B9F339__INCLUDED_

```

UC2View.cpp

```
// UC2View.cpp : implementation of the CUC2View class
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include <afxmt.h>

#include "UC2.h"

#include "UC2Doc.h"
#include "UC2View.h"
#include "MainFrm.h"
#include "ResMan.h"
#include "Stage.h"
#include "TileMap.h"
#include "Actor.h"
#include "UC2Ani/DIB.h"
#include "UC2Ani/PhSprite.h"
#include "UC2Ani/Sound.h"
#include "UC2Ani/MCIObj.h"
#include "EditSend.h"
#include "Parser.h"
#include "UC2CS.h"
#include "WhisperDlg.h"
#include "PPMemberInfo1.h"
#include "PSFrame.h"
// #include <hlink.h> // "urlmon.h" HlinkSimpleNavigateToString
#include "InputIntDlg.h"
#ifdef _MALL
#include "DlgPDA.h"
#endif

#include "httpUtility.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan    gResMan;
extern CParser    gParser;
const int  ACTOR_AREA_PIXEL=50; // cursor display
const DWORD TIMER_MSPT=1000L;
const DWORD PUNCH_MSPT=15*1000;

////////////////////////////////////
// CUC2View

IMPLEMENT_DYNCREATE(CUC2View, COSBView)

BEGIN_MESSAGE_MAP(CUC2View, COSBView)
   //{{AFX_MSG_MAP(CUC2View)
```

```

ON_WM_LBUTTONDOWN()
ON_WM_LBUTTONDBLCLK()
ON_WM_MOUSEMOVE()
ON_WM_RBUTTONDOWN()
ON_COMMAND(ID_VIEW_ADJUST_WINDOW, OnViewAdjustWindow)
ON_WM_SETFOCUS()
ON_WM_KEYDOWN()
ON_COMMAND(ID_ACTOR_VOICE, OnActorVoice)
ON_WM_DESTROY()
ON_WM_TIMER()
ON_COMMAND(ID_IGNORE, OnIgnore)
ON_COMMAND(ID_KICKOUT, OnKickout)
ON_COMMAND(ID_MAKEHOST, OnMakehost)
ON_COMMAND(ID_WHISPER, OnWhisper)
ON_UPDATE_COMMAND_UI(ID_VIEW_ADJUST_WINDOW, OnUpdateViewAdjustWindow)
ON_COMMAND(ID_ACTOR_HYPERLINK, OnActorHyperlink)
ON_COMMAND(ID_VIEW_BUBBLE_TEXTLIMIT, OnViewBubbleTextlimit)
ON_COMMAND(ID_VIEW_BUBBLE_TIME, OnViewBubbleTime)
//}}AFX_MSG_MAP
ON_COMMAND(ID_ACTOR_PROP, OnProperties)
ON_COMMAND_RANGE(ID_ACTOR_ACTION_0, ID_ACTOR_ACTION_9, OnActorAction)
ON_COMMAND_RANGE(ID_ACTOR_STATE_0, ID_ACTOR_STATE_2, OnActorState)
ON_COMMAND_RANGE(ID_ACTOR_MOVE_F, ID_ACTOR_MOVE_B, OnActorMove)
ON_COMMAND_RANGE(ID_ACTOR_TURN_L, ID_ACTOR_TURN_R, OnActorTurn)
// Standard printing commands
ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
// ON_NOTIFY(TTN_NEEDTEXT, 0, OnToolTipNotify)
END_MESSAGE_MAP()

/* MALL
ON_COMMAND(ID_VIEW_PDA, OnViewPda)
ON_COMMAND(ID_VIEW_IM, OnViewIm)
ON_COMMAND(ID_VIEW_TITANIC, OnViewTitanic)
ON_COMMAND(ID_VIEW_MAP, OnViewMap)
ON_COMMAND(ID_VIEW_PDA_MUSIC, OnViewPdaMusic)
*/
////////////////////////////////////
// CUC2View construction/destruction

CUC2View::CUC2View()
{
    TRACE0("CUC2View::CUC2View()\n");
    m_bFirstDraw      = TRUE;
    m_pActor           = NULL;
    m_uTimer           = 0;
    // for Stage
    m_dwChannelStartTick = 0L;
    m_dwSoundAlarmTick  = 0L;
    m_dwLastHitTick     = 0L;
    m_bVoiceCap         = FALSE;
    m_pPropFrame        = NULL;
}

CUC2View::~CUC2View()
{

```


UC2View.cpp

```

        TRACE0("CUC2View::~~CUC2View()\n");
    }

    BOOL CUC2View::PreCreateWindow(CREATESTRUCT& cs)
    {
        TRACE0("CUC2View::PreCreateWindow()\n");
        // TODO: Modify the Window class or styles here by modifying
        // the CREATESTRUCT cs

        return COSBView::PreCreateWindow(cs);
    }

    //////////////////////////////////////
    // CUC2View drawing

    void CUC2View::OnDraw(CDC* pDC)
    {
        // CUC2Doc* pDoc = GetDocument();
        // ASSERT_VALID(pDoc);

        if (m_bFirstDraw) // This is the moment when the user can apparently
            see this window.
        {
            TRACE0("CUC2View::OnDraw() - First call\n");
            m_bFirstDraw = FALSE;
            CEditSend* pES = GetDocument()->GetEditSend();
            if (pES)
                pES->SetFocus();
            // CSplashWnd::HideSplashScreen();
            // ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_COMMAND,
            (WPARAM) IDC_BTN_CREATE, (LPARAM) 0);
            #ifdef _MALL
                GetDocument()->ToggleDemo();
            #endif
        }
        COSBView::OnDraw(pDC);
    }

    void CUC2View::OnInitialUpdate()
    {
        TRACE0("CUC2View::OnInitialUpdate()\n");
        COSBView::OnInitialUpdate();

        CMainFrame* pMF = (CMainFrame*)GetParentFrame();
        TRACE("\n1");
        ASSERT(pMF);
        TRACE("\n2");
        CDIB* pDIB = GetOSB();
        TRACE("\n2");

        ASSERT(pDIB);
        TRACE("\n3");

        pMF->AdjustFrame(pDIB->GetWidth(), pDIB->GetHeight());
        TRACE("\n4");
    }

```

```

pMF->InitControlPanel((CPalette*)GetOSBPalette());
TRACE("\n5");

m_dwStageMSPT = ((CUC2App*)AfxGetApp())->RegGetStageSec();
m_dwStageMSPT *= 1000;
TRACE("m_dwStageMSPT = %ld\n", m_dwStageMSPT);
m_uTimer = SetTimer(4001, TIMER_MSPT, NULL);    // 1 sec. term

TRACE("\n6");

if (!m_uTimer)
{
    AfxMessageBox(IDS_ERROR_TIMER);
}
::SendMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_COMMAND,
(WPARAM) IDC_BTN_CREATE, (LPARAM) 0);
// EnableToolTips();
}

////////////////////////////////////
// CUC2View printing

BOOL CUC2View::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CUC2View::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CUC2View::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

////////////////////////////////////
// CUC2View diagnostics

#ifdef _DEBUG
void CUC2View::AssertValid() const
{
    COSBView::AssertValid();
}

void CUC2View::Dump(CDumpContext& dc) const
{
    COSBView::Dump(dc);
}

CUC2Doc* CUC2View::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CUC2Doc)));
    return (CUC2Doc*)m_pDocument;
}

```

```

#endif // _DEBUG

/////////////////////////////////////////////////////////////////
// CUC2View message handlers
/////////////////////////////////////////////////////////////////
// Frame

// Adjust the frame window to be fit to this OSB
void CUC2View::AdjustFrameWindow()
{
    CMainFrame* pMF = (CMainFrame*)GetParentFrame();
    ASSERT(pMF);
    if (GetDocument()->IsHistoryPanel())
        return;
    CDIB* pDIB = GetOSB();
    ASSERT(pDIB);
    pMF->AdjustFrame(pDIB->GetWidth(), pDIB->GetHeight());
}

/////////////////////////////////////////////////////////////////
// OSB: Off-Screen Buffered View
/////////////////////////////////////////////////////////////////

// Create a new buffer and palette to match a new background DIB
// virtual in COSBView
BOOL CUC2View::CreateOSB(CDIB* pDIB)
{
    // m_pCS = NULL;
    // m_pMS = NULL;

    // Create a new buffer and palette
    if (!COSBView::CreateOSB(pDIB))
        return FALSE;

    // Map the colors of the background DIB to
    // the identity palette we just created for the background
    pDIB->MapColorsToPalette((CPalette*)GetOSBPalette());

    // Resize the main frame window to fit the background image
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit(FALSE); // Try shrinking first
    ResizeParentToFit(TRUE);  // Let's be daring

    Render(); // Render the entire scene to the off-screen buffer
    // DrawOSB(); // Paint the off-screen buffer to the window
    Invalidate(FALSE);

    return TRUE;
}

// Render the scene to the off-screen buffer pClipRect defaults to NULL
// virtual in COSBView
void CUC2View::Render(CRect* pClipRect)
{
    CStage* pStage = GetDocument()->GetStage();
    if (pStage)
        pStage->Render(pClipRect);
}

```

```

}

void CUC2View::NewStageLoaded()
{
    m_dwChannelStartTick = m_dwSoundAlarmTick = ::GetTickCount();
}

////////////////////////////////////
// User Interface
void CUC2View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
    CPoint lpt(point);
    dc.DPtoLP(&lpt);

    CStage* pStage = GetDocument()->GetStage();
    if (!pStage)
        return;

    // CPhasedSprite* pPS = pStage->AnimatedHitTest(lpt);
    // Assume that we have not CSprite, we have only CPhasedSprite and
derived...
    CPhasedSprite* pPS = (CPhasedSprite*)pStage->SpriteHitTest(lpt);
    if (pPS && pPS->HasHyperlink() &&
        (pPS->GetLinkType() == CPhasedSprite::HLINK_HTTP))
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_HGREP_DOWN));
        int n = pPS->GetCellID();
        gParser.CopyBuffer(*pPS->GetHyperlink());
        CString strHL;
        while (gParser.GetValueRightToken(strHL, '|') && n--)
            ;

        if (!strHL.IsEmpty())
        {
            ((CMainFrame*)AfxGetMainWnd())->ShellBrowseURL(strHL);
            return;
        }
    }

    CActor* pA = pStage->GetThisActor();
    if (pA)
    {
        CRect rcA2;
        pA->GetRect(rcA2);
        rcA2.InflateRect(ACTOR_AREA_PIXEL, ACTOR_AREA_PIXEL);
        if (!rcA2.PtInRect(lpt))
        {
            ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
            return;
        }
        CPoint ptA(pA->GetCenter());
        WORD wDA = pA->GetDA();
    }
}

```

```

WORD wDACursor;
if (lpt.y < ptA.y)      // Backward
{
    if (lpt.x < ptA.x)    // DA_BL
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_ARROW_LT));
        wDACursor = DA_BL;
    }
    else // DA_BR
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_ARROW_RT));
        wDACursor = DA_BR;
    }
}
else // Forward
{
    if (lpt.x < ptA.x)    // DA_FR
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_ARROW_LB));
        wDACursor = DA_FR;
    }
    else // DA_FL
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_ARROW_RB));
        wDACursor = DA_FL;
    }
}

if (wDACursor == wDA)
    OnActorMove(ID_ACTOR_MOVE_F);
else
{
    int nID;
    switch (wDACursor)
    {
        case DA_FR: nID = (wDA == DA_BL) ? ID_ACTOR_TURN_L :
ID_ACTOR_TURN_R; break;
        case DA_FL: nID = (wDA == DA_FR) ? ID_ACTOR_TURN_L :
ID_ACTOR_TURN_R; break;
        case DA_BR: nID = (wDA == DA_FL) ? ID_ACTOR_TURN_L :
ID_ACTOR_TURN_R; break;
        case DA_BL: nID = (wDA == DA_BR) ? ID_ACTOR_TURN_L :
ID_ACTOR_TURN_R; break;
        default:    nID = ID_ACTOR_TURN_L;
    }
    OnActorTurn(nID);
}
else
{
    ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
}
// COSBView::OnLButtonDown(nFlags, point);
}

void CUC2View::OnLButtonDblClk(UINT nFlags, CPoint point)
{

```

```

        CClientDC dc(this);
        OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
        CPoint lpt(point);
        dc.DPtoLP(&lpt);
/*
        CUC2Doc* pDoc = GetDocument();
        CStage* pStage = pDoc->GetStage();
        if (!pStage)
            return;
// For Demo
        static int nChar=0;
        CActor* pActor = pStage->CreateActor(nChar, lpt, FALSE); // AS_STAND |
DA_FR, (nChar%4 == 0));
        pActor->Act(CMD_STAND + nChar % 24);
        if (++nChar >= 18)
            nChar = 0;
        m_pActor = pActor;

        RenderAndDrawDirtyList();
*/
// COSBView::OnLButtonDblClk(nFlags, point);
}

void CUC2View::OnMouseMove(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
    CPoint lpt(point);
    dc.DPtoLP(&lpt);

    CStage* pStage = GetDocument()->GetStage();
    if (!pStage)
        return;

//    CPhasedSprite* pPS = pStage->AnimatedHitTest(lpt);
    CPhasedSprite* pPS = (CPhasedSprite*)pStage->SpriteHitTest(lpt);
    if (pPS && pPS->HasHyperlink() &&
        (pPS->GetLinkType() == CPhasedSprite::HLINK_HTTP))
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_HGREP_UP));
        return;
    }

    CActor* pA = pStage->GetThisActor();
    if (pA)
    {
        CRect rcA;
        pA->GetRect(rcA);
        CRect rcA2(rcA);
        rcA2.InflateRect(ACTOR_AREA_PIXEL, ACTOR_AREA_PIXEL);
        if (!rcA2.PtInRect(lpt) || rcA.PtInRect(lpt))
        {
            ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
            return;
        }
    }
}

```

```

        CPoint ptA(pA->GetCenter());
        if (lpt.x < ptA.x)
        {
            ::SetCursor(AfxGetApp()->LoadCursor((lpt.y < ptA.y) ?
                IDC_ARROW_LT : IDC_ARROW_LB));
        }
        else
        {
            ::SetCursor(AfxGetApp()->LoadCursor((lpt.y < ptA.y) ?
                IDC_ARROW_RT : IDC_ARROW_RB));
        }
    }
    else
    {
        ::SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
    }
}
// COSBView::OnMouseMove(nFlags, point);
}

void CUC2View::OnRButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
    CPoint lpt(point);
    dc.DPtoLP(&lpt);

    CUC2Doc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;

    CPhasedSprite* pPS = pStage->AnimatedHitTest(lpt);
    if (!pPS || (pPS->GetSrcType() != SPRITE_ACTOR))
        return;
    // ASSERT(pPS->IsKindOf(RUNTIME_CLASS(CActor)));
    m_pActor = (CActor*)pPS;

    BOOL bThis = (m_pActor == pDoc->GetThisActor());
    CMenu menu;
    VERIFY(menu.LoadMenu(bThis ? IDR_MENU_ACTOR : IDR_MENU_ACTOR_OTHER));
    CMenu* pPopup = menu.GetSubMenu(0);
    ASSERT(pPopup);

    CString strVer;
    int nVer = m_pActor->m_mi.GetVersion();
    strVer.Format(" V%d.%02d", nVer / 100, nVer % 100);
    CString strInfo(*(m_pActor->m_mi.GetNick()) + _T("(") + *(m_pActor-
>m_mi.GetRealName()) + _T(")"));
    if (strInfo.GetLength() > 30)
    {
        strInfo.ReleaseBuffer(30);
    }
    strInfo += strVer;
    if (m_pActor->IsMemberHost())
        strInfo += "<HOST>";
    pPopup->ModifyMenu(ID_ACTOR_PROP, MF_BYCOMMAND | MF_STRING,

```

```

ID_ACTOR_PROP, strInfo);

if (HasVoiceCap())
{
    if (m_pActor->IsVoice())
        pPopup->CheckMenuItem(ID_ACTOR_VOICE, MF_CHECKED);
}
else
{
    pPopup->EnableMenuItem(ID_ACTOR_VOICE, MF_GRAYED);
}

if (bThis)
{
    switch (m_pActor->GetState() & AS_MASK)
    {
        case AS_STAND:
            pPopup->CheckMenuItem(ID_ACTOR_STATE_0, MF_CHECKED);
            break;
        case AS_MORPH:
            {
                pPopup->CheckMenuItem(ID_ACTOR_STATE_1, MF_CHECKED);
                for (int i=ID_ACTOR_ACTION_0; i < ID_ACTOR_ACTION_7; i++)
                    pPopup->EnableMenuItem(ID_ACTOR_ACTION_0, MF_GRAYED);
            }
            break;
        case AS_DOZE:
            pPopup->CheckMenuItem(ID_ACTOR_STATE_2, MF_CHECKED);
            break;
    }
}
else
{
    if (m_pActor->IsMemberIgnored())
        pPopup->CheckMenuItem(ID_IGNORE, MF_CHECKED);

    CUC2Channel* pChannel = pDoc->GetChannel();
    if (pChannel && !pChannel->FAMHost())
    {
        pPopup->EnableMenuItem(ID_KICKOUT, MF_GRAYED);
        pPopup->EnableMenuItem(ID_MAKEHOST, MF_GRAYED);
    }
    if (m_pActor->IsMemberHost())
        pPopup->CheckMenuItem(ID_MAKEHOST, MF_CHECKED);
    pPopup->EnableMenuItem(ID_ACTOR_HYPERLINK,
        m_pActor->m_mi.HasHyperlink() ? MF_CHECKED :
MF_GRAYED);
}

ClientToScreen(&point);
pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
point.y, this);
// COSBView::OnRButtonDown(nFlags, point);
}

void CUC2View::OnViewAdjustWindow()
{

```



```

        AdjustFrameWindow();
    }

void CUC2View::OnUpdateViewAdjustWindow(CCmdUI* pCmdUI)
{
    GetDocument()->IsHistoryPanel();
}

void CUC2View::OnSetFocus(CWnd* pOldWnd)
{
    COSBView::OnSetFocus(pOldWnd);

    if (m_bFirstDraw) // not initialized yet
        return;
    CEditSend* pES = GetDocument()->GetEditSend();
    if (pES)
        pES->SetFocus();
}
/*
int CUC2View::OnToolHitTest(CPoint point, TOOLINFO* pTI) const
{
    CRect rc(0, 0, 300, 300);
    if (rc.PtInRect(point))
    {
        ASSERT(pTI);
        pTI->hwnd = m_hWnd;
        pTI->rect = rc;
        pTI->uId = IDC_BTN_CREATE;
        pTI->lpszText = LPSTR_TEXTCALLBACK;
        return 1;
    }
    return CView::OnToolHitTest(point, pTI);
}

void CUC2View::OnToolTipNotify(UINT id, NMHDR* pNMH, LRESULT* pResult)
{
    TOOLTIPTEXT* pTTT = (TOOLTIPTEXT*)pNMH;
    UINT nID = pNMH->idFrom;
    if (pTTT->uFlags & TTF_IDISHWND)
    {
        nID = ::GetDlgCtrlID((HWND)nID);
        ASSERT(nID);
    }
    pTTT->lpszText = MAKEINTRESOURCE(nID);
    pTTT->hinst = AfxGetResourceHandle();
}
*/

BOOL CUC2View::PunchAvailable() const
{
    return ((::GetTickCount() - m_dwLastHitTick) >= PUNCH_MSPT);
}

void CUC2View::OnActorAction(UINT nID)
{
    CUC2Doc* pDoc = GetDocument();
    pDoc->SendCommand(CMD_SMILE + nID - ID_ACTOR_ACTION_0);
}

```

```

    /**/ For Demo
        if (!m_pActor)
            return;
        m_pActor->Act(CMD_SMILE + nID - ID_ACTOR_ACTION_0);
    */
}

void CUC2View::OnActorState(UINT nID)
{
    CUC2Doc* pDoc = GetDocument();
    pDoc->SendCommand(CMD_STAND + nID - ID_ACTOR_STATE_0);
    /**/ For Demo
        if (!m_pActor)
            return;
        m_pActor->Act(CMD_STAND + nID - ID_ACTOR_STATE_0);
    */
}

void CUC2View::OnActorMove(UINT nID)
{
    CUC2Doc* pDoc = GetDocument();
    if (pDoc->IsDemo())
        return;
    pDoc->SendMoveCommand(CMD_MOVEF + nID - ID_ACTOR_MOVE_F);
    /**/ For Demo
        if (!m_pActor)
            return;

        CStage* pStage = GetDocument()->GetStage();
        if (!pStage)
            return;
        pStage->ActorMove(m_pActor, nID == ID_ACTOR_MOVE_F);
    */
}

void CUC2View::OnActorTurn(UINT nID)
{
    CUC2Doc* pDoc = GetDocument();
    pDoc->SendCommand(CMD_TURNL + nID - ID_ACTOR_TURN_L);
    /**/ For Demo
        if (!m_pActor)
            return;
        m_pActor->Act(CMD_TURNL + nID - ID_ACTOR_TURN_L);
    */
}

void CUC2View::OnActorVoice()
{
    if (!m_pActor)
        return;
}

void CUC2View::OnDestroy()
{
    COSBView::OnDestroy();
    if (m_uTimer)
        KillTimer(m_uTimer);
}

```

```

}

void CUC2View::OnTimer(UINT nIDEvent)
{
    CUC2Doc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;

    DWORD dwCurTick = ::GetTickCount();
    if (dwCurTick > m_dwSoundAlarmTick)
    {
        if (pDoc->IsSoundOn())
            pStage->PlayBGM();

        DWORD dwRand = (DWORD)(1*90*1000 + rand() % (3*60*1000));
        m_dwSoundAlarmTick = dwCurTick + dwRand;
        TRACE("Music play queued after %.2f min.\n",
            float(m_dwSoundAlarmTick - dwCurTick)/(60*1000L));
    }
    if (!pStage->IsExitOpen() &&
        (dwCurTick - m_dwChannelStartTick) > m_dwStageMSPT)
    {
        pStage->SetExitOpen(); // Open passages to other channels
    }
/*
    if (pDoc->GetUC2Mode() & UC2MODE_DEMO)
    {
        UC2Demo();
    }
*/
// COSBView::OnTimer(nIDEvent);
}

void CUC2View::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
/*
    CUC2Doc* pDoc = GetDocument();
    if (pDoc->GetThisActor() == NULL)
        return;
    // switch for input key
    switch (nChar)
    {
        case VK_UP:      pDoc->SendCommand(CMD_MOVE_NORTH); break;
        case VK_LEFT:    pDoc->SendCommand(CMD_MOVE_WEST); break;
        case VK_DOWN:    pDoc->SendCommand(CMD_MOVE_SOUTH); break;
        case VK_RIGHT:   pDoc->SendCommand(CMD_MOVE_EAST); break;
        case VK_F5:      SetMode(0); break;
        case VK_F6:      SetMode(1); break;
        case VK_F7:      SetMode(1); break;
        case VK_F8:      SetMode(2); break;
        case VK_F9:      SetMode(3); break;
        case VK_F10:     SetMode(4); break;
    }
    // Show current state in the status bar
    CMainFrame* pFrame = (CMainFrame*)(AfxGetApp()->m_pMainWnd);
    ASSERT(pFrame);
    pFrame->m_wndStatusBar.SetWindowText(buf);
*/
}

```

```

*/    COSBView::OnKeyDown(nChar, nRepCnt, nFlags);
}

void CUC2View::OnIgnore()
{
    if (!m_pActor)
        return;
    PICS_MEMBER pM = m_pActor->m_mi.GetMember();
    if (!pM)
        return;
    BOOL bIgnored = (pM->HrIsMemberIgnored() == NOERROR);
    bIgnored = !bIgnored;
    pM->HrSetIgnoreMember(bIgnored);    // returns NOERROR if successful
    m_pActor->Act(bIgnored ? CMD_DOZE : CMD_STAND);
    // GetDocument()->WhisperTo(m_pActor, _T("«½Ã ÁßÄÖ`İ`Ü."));
    pM->Release();
}

void CUC2View::OnKickout()
{
    if (!m_pActor)
        return;
    PICS_MEMBER pM = m_pActor->m_mi.GetMember();
    if (!pM)
        return;
    CWhisperDlg dlg;
    dlg.m_strToID = _T("To ") + *m_pActor->m_mi.GetNick();
    if ((dlg.DoModal() == IDOK) && !dlg.m_strText.IsEmpty())
    {
        int len = dlg.m_strText.GetLength();
        char* pSz = dlg.m_strText.GetBuffer(len);
        pM->HrCloseA(pSz);
    }
    pM->Release();
}

void CUC2View::OnMakehost()
{
    if (!m_pActor)
        return;
    PICS_MEMBER pM = m_pActor->m_mi.GetMember();
    if (!pM)
        return;
    pM->HrMakeHost();
    pM->Release();
}

void CUC2View::OnWhisper()
{
    if (!m_pActor)
        return;
    CWhisperDlg dlg;
    dlg.m_strToID = *m_pActor->m_mi.GetNick();
    if ((dlg.DoModal() == IDOK) && !dlg.m_strText.IsEmpty())
    {
        GetDocument()->WhisperTo(m_pActor, dlg.m_strText);
    }
}

```

```

}

void CUC2View::OnProperties()
{
    if (!m_pActor)
        return;

    if (!m_pPropFrame)
    {
        m_pPropFrame = new CPSFrame;
        CRect rect(0, 0, 0, 0);
        CString strTitle;
        VERIFY(strTitle.LoadString(IDS_ACTOR_PROPSHT_CAPTION));

        if (!m_pPropFrame->Create(NULL, strTitle, WS_POPUP | WS_CAPTION |
WS_SYSMENU, rect, this))
        {
            delete m_pPropFrame;
            m_pPropFrame = NULL;
            return;
        }
        m_pPropFrame->CenterWindow();
    }

    if (m_pPropFrame) // After creation
    {
        // Before un hiding the modeless property sheet, update its
        settings appropriately.
        // For example, if you are reflecting the state of the currently
        selected item,
        // pick up that information from the active view and change the
        property sheet settings now.
        CPPMemberInfo1& page1 = m_pPropFrame->m_pModelessPropSheet->
>m_Page1;
        page1.m_strNick = *m_pActor->m_mi.GetNick();
        page1.m_strUserID = *m_pActor->m_mi.GetUserID();
        page1.m_strRealName = *m_pActor->m_mi.GetRealName();
        page1.m_strSexAge.Format("%s %d", (m_pActor->m_mi.GetSex() == 0)
? _T("M") : _T("F"),
                                m_pActor->m_mi.GetAge());
        int nVer = m_pActor->m_mi.GetVersion();
        page1.m_strVersion.Format("%d.%02d", nVer / 100, nVer % 100);
        page1.UpdateData(FALSE);
        m_pPropFrame->m_pModelessPropSheet->SetActivePage(&page1);

        CPPMemberInfo2& page2 = m_pPropFrame->m_pModelessPropSheet->
>m_Page2;
        page2.m_strProfile = *m_pActor->m_mi.GetProfile();

        if (!m_pPropFrame->IsWindowVisible())
            m_pPropFrame->ShowWindow(SW_SHOW);
    }
}

void CUC2View::OnActorHyperlink()
{
    if (m_pActor && m_pActor->m_mi.HasHyperlink())

```

```

    {
        CMainFrame* pMF = (CMainFrame*)GetParentFrame();
        ASSERT(pMF);
        pMF->ShellBrowseURL(*m_pActor->m_mi.GetHyperlink());
    }
}

void CUC2View::OnViewBubbleTextlimit()
{
    CInputIntDlg dlg;
    CString strTitle;
    strTitle.LoadString(IDS_ENTER_BUBBLE_TEXT_LIMIT);
    dlg.SetTitle(strTitle);
    CUC2App* pApp = (CUC2App*)AfxGetApp();
    dlg.m_nVal = pApp->RegGetBubbleTextLength();
    dlg.SetDefaultVal(gResMan.GetDefaultBubbleTextLimit());
    if (dlg.DoModal() == IDOK)
    {
        if (dlg.m_nVal < 0)
            dlg.m_nVal = 0;
        gResMan.SetBubbleTextLimit(dlg.m_nVal);
        pApp->RegSetBubbleTextLength(dlg.m_nVal);
    }
}

void CUC2View::OnViewBubbleTime()
{
    CInputIntDlg dlg;
    CString strTitle;
    strTitle.LoadString(IDS_ENTER_BUBBLE_TIME);
    dlg.SetTitle(strTitle);
    CUC2App* pApp = (CUC2App*)AfxGetApp();
    dlg.m_nVal = pApp->RegGetBubbleTime()/1000;
    dlg.SetDefaultVal(gResMan.GetDefaultBubbleTime()/1000);
    if (dlg.DoModal() == IDOK)
    {
        if (dlg.m_nVal < 0)
            dlg.m_nVal = 0;
        gResMan.SetBubbleTime(dlg.m_nVal*1000);
        pApp->RegSetBubbleTime(gResMan.GetBubbleTime());
    }
}

#ifdef _MALL
void CUC2View::OnViewPda()
{
    CDlgPDA pda("MPAniU.bmp", "MPAniL.bmp", 6);
    pda.SetMode(TRUE, TRUE);
    if (pda.DoModal() == IDOK)
    {
    }
}

void CUC2View::OnViewPdaMusic()
{
    CDlgPDA pda("MPAniUM.bmp", "MPAniLM.bmp", 4);
    pda.SetMode(TRUE, TRUE);
}

```

UC2View.cpp

```
        if (pda.DoModal() == IDOK)
        {
        }
    }

void CUC2View::OnViewIm()
{
    CUC2Doc* pDoc = GetDocument();
    pDoc->ShowIMDemo();
}

void CUC2View::OnViewTitanic()
{
    CUC2Doc* pDoc = GetDocument();
    pDoc->ShowTitanicDemo();
}

void CUC2View::OnViewMap()
{
    CUC2Doc* pDoc = GetDocument();
    pDoc->ShowMapDemo();
}
#endif
```

UC2View.h

```
// UC2View.h : interface of the CUC2View class
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT NETWORK INC
//=====

#ifndef AFX_UC2VIEW_H_A1313871_A610_11D1_80E2_080009B9F339__INCLUDED_
#define AFX_UC2VIEW_H_A1313871_A610_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "UC2Ani/OSBView.h"

class CActor;
class CPSFrame;

class CUC2View : public COSBView
{
protected: // create from serialization only
    CUC2View();
    DECLARE_DYNCREATE(CUC2View)

// Attributes
public:
    CUC2Doc* GetDocument();
    BOOL HasVoiceCap() const { return m_bVoiceCap; }

// Operations
public:
    virtual BOOL CreateOSB(CDIB* pDIB); // COSBView
    virtual void Render(CRect* pClipRect=NULL); // COSBView
    void AdjustFrameWindow();
    void NewStageLoaded();
    void SetActor(CActor* pA) { m_pActor = pA; }
    BOOL PunchAvailable() const;
    void SaveLastHitTick() { m_dwLastHitTick =
::GetTickCount(); }

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUC2View)
public:
    virtual void OnDraw(CDC* pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnInitialUpdate();
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
```



```

    virtual ~CUC2View();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
//    int          OnToolHitTest(CPoint point, TOOLINFO* pTI) const;
    void          OnProperties();

private:
    CActor*       m_pActor;    // pointer to current sprite (menu)
    BOOL          m_bFirstDraw;
    UINT          m_uTimer;
    DWORD         m_dwChannelStartTick;    // Ticks on starting this channel
    DWORD         m_dwSoundAlarmTick;
    DWORD         m_dwStageMSPT;           // Delay in a stage
    DWORD         m_dwLastHitTick;        // Last punch
    BOOL          m_bVoiceCap;
    CPSFrame*     m_pPropFrame;

// Generated message map functions
protected:
   //{{AFX_MSG(CUC2View)
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnViewAdjustWindow();
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
    afx_msg void OnActorVoice();
    afx_msg void OnDestroy();
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg void OnIgnore();
    afx_msg void OnKickout();
    afx_msg void OnMakehost();
    afx_msg void OnWhisper();
    afx_msg void OnUpdateViewAdjustWindow(CCmdUI* pCmdUI);
    afx_msg void OnActorHyperlink();
    afx_msg void OnViewBubbleTextlimit();
    afx_msg void OnViewBubbleTime();
   //}}AFX_MSG
    afx_msg void OnActorAction(UINT nID);
    afx_msg void OnActorState(UINT nID);
    afx_msg void OnActorMove(UINT nID);
    afx_msg void OnActorTurn(UINT nID);
//    void OnToolTipNotify(UINT id, NMHDR* pNMH, LRESULT* pResult);
    DECLARE_MESSAGE_MAP()
};

/* MALL
    afx_msg void OnViewPda();
    afx_msg void OnViewIm();
    afx_msg void OnViewTitanic();
    afx_msg void OnViewMap();
    afx_msg void OnViewPdaMusic();

```

UC2View.h

*/

```
#ifndef _DEBUG // debug version in UC2View.cpp
inline CUC2Doc* CUC2View::GetDocument()
{ return (CUC2Doc*)m_pDocument; }
#endif
```

////////////////////////////////////

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
```

```
#endif //
```

```
#ifndef(AFX_UC2VIEW_H__A1313871_A610_11D1_80E2_080009B9F339__INCLUDED_)
```

```

// UCServerSession.cpp : implementation of the CUCServerSession class
//
//=====
//      (C) Programmed by Satya Sudhakar M, Apr 2000
//      UNICHAT NETWORKS
//=====

#include "stdafx.h"
#include "UC2.h"
#include "UCServerSession.h"
#include <afxinet.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define ISNULL(X) (X==NULL?TRUE:FALSE)

IMPLEMENT_DYNCREATE(CUCServerSession, CInternetSession)

CUCServerSession::CUCServerSession() : CInternetSession()
{
    m_URL = _T("http://www.unichat.com/ads/index.html");
    m_pUrlConnection = NULL;
    m_pFTPConnection = NULL;
    m_pHTTPConnection = NULL;
    m_pGopherConnection = NULL;
}

CUCServerSession::~CUCServerSession()
{
}

#ifdef _DEBUG
void CUCServerSession::AssertValid() const
{
    CInternetSession::AssertValid();
}

void CUCServerSession::Dump(CDumpContext& dc) const
{
    CInternetSession::Dump(dc);
}
#endif // _DEBUG

CStdioFile* CUCServerSession::OpenURL( LPCTSTR pstrURL, DWORD dwContext ,
DWORD dwFlags , LPCTSTR pstrHeaders , DWORD dwHeadersLength )
{

```

```

        m_pUrlConnection = CInternetSession ::
OpenURL(pstrURL,dwContext,dwFlags,pstrHeaders,dwHeadersLength);
        return m_pUrlConnection;

}

BOOL CUCServerSession :: MakeFTTPConnection(LPCTSTR pstrServer)
{
    if (ISNULL(pstrServer))
        return FALSE;
    m_pFTPConnection = CInternetSession ::GetFtpConnection(pstrServer);
    if (ISNULL(m_pFTPConnection))
        return FALSE;
    return TRUE;
}

BOOL CUCServerSession :: DownloadfileFromServer(LPCTSTR
FilePathOnserver,LPCTSTR FileLocalPath)
{
    if (ISNULL (FileLocalPath) || ISNULL(FileLocalPath))
        return FALSE;
    if (ISNULL(m_pFTPConnection))
    {
        m_pFTPConnection = CInternetSession ::GetFtpConnection(m_URL);
        if (ISNULL (m_pFTPConnection))
            return FALSE;
    }
    return m_pFTPConnection->GetFile(FilePathOnserver,FileLocalPath);
}

```

UCServerSession.h

```
#ifndef UCSEVERSESSION_H
#define UCSEVERSESSION_H

// File name : UCServerSession.h : header file
// Created by : Satya Sudhakar Mukkamala
// Created UCServerSession.h class for ServerConnection
////////////////////////////////////
// CUCServerSession

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif
#include <afxinet.h>

class CUCServerSession : public CInternetSession
{
public:
    CUCServerSession();
    virtual ~CUCServerSession();
    DECLARE_DYNCREATE(CUCServerSession)

public:
    //{AFX_DATA(CUCServerSession)
    // NOTE: the ClassWizard will add data members here
    //}AFX_DATA

// Attributes
public:
    CString GetURL()
    {
        return m_URL;
    }
    CStdioFile* GetUrlConnection()
    {
        return m_pUrlConnection;
    }
    CFTPConnection* GetFTPConnection()
    {
        return m_pFTPConnection;
    }
    CHttpConnection* GetHTTPConnection()
    {
        return m_pHTTPConnection;
    }
    CGopherConnection* GetGopherConnection()
    {
        return m_pGopherConnection;
    }
    void SetURL(CString url)
    {
        m_URL = url;
    }
}
```

```

    }

// Operations
public:
    CStdioFile* OpenURL( LPCTSTR pstrURL, DWORD dwContext = 1, DWORD
dwFlags = INTERNET_FLAG_TRANSFER_ASCII, LPCTSTR pstrHeaders = NULL, DWORD
dwHeadersLength = 0 );
    BOOL MakeFTPConnection(LPCTSTR pstrServer);
    BOOL DownloadfileFromServer(LPCTSTR FilePathOnserver,LPCTSTR
FileLocalPath);
// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CUCServerSession)
    //}AFX_VIRTUAL

// Implementation
protected:
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
private:
    CString m_URL;
    CStdioFile* m_pUrlConnection;
    CFTPConnection* m_pFTPConnection;
    CHttpConnection* m_pHTTPConnection;
    CGopherConnection* m_pGopherConnection;

public:
    // Generated message map functions
    //{AFX_MSG(CUCServerSession)
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif

```

UCSplitterWnd.cpp

```
// UCSplitterWnd.cpp : implementation of the CUCSplitterWnd class
//
//=====
//      (C) Programmed by Satya Sudhakar M, Apr 2000
//      UNICHAT NETWORKS
//=====

#include "stdafx.h"
#include "UC2.h"
#include "UCSplitterWnd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CUCSplitterWnd, CSplitterWnd)

// Message map for CUCSplitterWnd
BEGIN_MESSAGE_MAP(CUCSplitterWnd, CSplitterWnd)
//{{AFX_MSG_MAP(CUCSplitterWnd)
// NOTE - the ClassWizard will add and remove mapping macros
here.
//      DO NOT EDIT what you see in these blocks of generated code
!
        ON_WM_LBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

CUCSplitterWnd::CUCSplitterWnd() : CSplitterWnd()
{
}

CUCSplitterWnd::~CUCSplitterWnd()
{
}

#ifdef _DEBUG
void CUCSplitterWnd::AssertValid() const
{
    CSplitterWnd::AssertValid();
}

void CUCSplitterWnd::Dump(CDumpContext& dc) const
{
    CSplitterWnd::Dump(dc);
}
#endif // _DEBUG

BOOL CUCSplitterWnd::Create(CWnd* pParentWnd, int nMaxRows, int nMaxCols,
SIZE sizeMin, CCreateContext* pContext, DWORD dwStyle, UINT nID)
{

```

```
        return CSplitterWnd :: Create(pParentWnd, nMaxRows, nMaxCols,
sizeMin, pParentWnd,dwStyle , nID);
    }

    BOOL CUCSplitterWnd ::CreateStatic( CWnd* pParentWnd, int nRows, int nCols,
    DWORD dwStyle, UINT nID )
    {
        return CSplitterWnd ::CreateStatic(pParentWnd,nRows,nCols,dwStyle, nID
    );
    }

    BOOL CUCSplitterWnd :: CreateCommon(CWnd* pParentWnd,SIZE sizeMin, DWORD
    dwStyle, UINT nID)
    {
        return CSplitterWnd :: CreateCommon(pParentWnd, sizeMin,dwStyle, nID);
    }

    void CUCSplitterWnd::OnLButtonDown(UINT nFlags, CPoint point)
    {
        // Just return it don't call the baseclass function
        // we don't allow to resize the SplitterWindow.
        return;
    }
```


UCSplitterWnd.h

```
#ifndef UCSPLITTER_H
#define UCSPLITTER_H

// File name : UCSplitterWnd.h : header file
// Created by : Satya Sudhakar Mukkamala
// Created CUCSplitterWnd class for Splitter Window.
////////////////////////////////////
// CUCSplitterWnd CSplitterWnd

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif
#include <afxhtml.h>

class CUCSplitterWnd : public CSplitterWnd
{
public:
    CUCSplitterWnd();           // protected constructor used by dynamic
creation
    virtual ~CUCSplitterWnd();
    DECLARE_DYNCREATE(CUCSplitterWnd)

public:
    //{AFX_DATA(CWebView)
    // NOTE: the ClassWizard will add data members here
    //}AFX_DATA

    // Attributes
public:

    // Operations
public:
    BOOL Create( CWnd* pParentWnd, int nMaxRows, int nMaxCols, SIZE sizeMin,
CCreateContext* pContext, DWORD dwStyle = WS_CHILD | WS_VISIBLE | WS_HSCROLL |
WS_VSCROLL | SPLS_DYNAMIC_SPLIT, UINT nID = AFX_IDW_PANE_FIRST );
    BOOL CreateStatic( CWnd* pParentWnd, int nRows, int nCols, DWORD dwStyle =
WS_CHILD | WS_VISIBLE, UINT nID = AFX_IDW_PANE_FIRST );
    BOOL CreateCommon(CWnd* pParentWnd, SIZE sizeMin, DWORD dwStyle, UINT nID);

    // Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CUCSplitterWnd)
    //}AFX_VIRTUAL

    // Implementation
protected:
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
public:
    // Generated message map functions
```

UCSplitterWnd.h

```
//{{AFX_MSG(CUCSplitterWnd)
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
```

WebView.cpp

```
// WebView.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "WebView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// WebView

IMPLEMENT_DYNCREATE(CWebView, CHtmlView)

CWebView::CWebView()
{
    TRACE0("CWebView::CWebView()");

    //{AFX_DATA_INIT(CWebView)
    // NOTE: the ClassWizard will add member initialization here
    //}AFX_DATA_INIT
}

CWebView::~CWebView()
{
}

void CWebView::DoDataExchange(CDataExchange* pDX)
{
    CHtmlView::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CWebView)
    // NOTE: the ClassWizard will add DDX and DDV calls here
    //}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CWebView, CHtmlView)
    //{AFX_MSG_MAP(CWebView)
    ON_WM_CREATE()
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// WebView diagnostics

#ifdef _DEBUG
void CWebView::AssertValid() const
{
    CHtmlView::AssertValid();
}

void CWebView::Dump(CDumpContext& dc) const
```

WebView.cpp

```
{
    CHtmlView::Dump(dc);
}
#endif //_DEBUG

void CWebView::OnInitialUpdate()
{
    TRACE("CWebView::OnInitialUpdate()");
    //TODO: This code navigates to a popular spot on the web.
    //Change the code to go where you'd like.
    SetScrollSizes(MM_TEXT, CSize(0, 0), CSize(0,0), CSize(0,0));
    Navigate2(_T("http://www.unichat.com/ads/index.html"), NULL, NULL);
}

void CWebView::OnGoBannerAd()
{
    Navigate2(_T("http://www.unichat.com/ads/index.html"), NULL, NULL);
}

/*void CWebView::OnGoDatingMatchcom()
{
    Navigate2(_T("http://www.match.com"), NULL, NULL);
}

void CWebView::OnGoMusicMusicblvd()
{
    Navigate2(_T("http://www.musicblvd.com"), NULL, NULL);
}

void CWebView::OnGoSoftwareBeyondsw()
{
    Navigate2(_T("http://www.beyond.com"), NULL, NULL);
}

void CWebView::OnGoVideoBigstar()
{
    Navigate2(_T("http://www.bigstar.com"), NULL, NULL);
}

void CWebView::OnGoWebsearchExcite()
{
    Navigate2(_T("http://www.excite.com"), NULL, NULL);
}
```

```

void CWebView::OnGoWebsearchInfoseek()
{
    Navigate2(_T("http://www.infoseek.com"), NULL, NULL);
}

void CWebView::OnGoWebsearchYahoo()
{
    Navigate2(_T("http://www.yahoo.com"), NULL, NULL);
}
*/

void CWebView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    TRACE("\nCWebView::OnUpdate()");
    OnGoBannerAd();
}

int CWebView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    //lpCreateStruct->
    if (CHtmlView::OnCreate(lpCreateStruct) == -1)
        return -1;

    return 0;
}

BOOL CWebView::PreCreateWindow(CREATESTRUCT& cs)
{
    return CHtmlView::PreCreateWindow(cs);
}

```

WebView.h

```
#if !defined(AFX_CWEBVIEW_H_EC3381DA_D9CA_11D2_A30A_00105A60F930__INCLUDED_)
#define AFX_CWEBVIEW_H_EC3381DA_D9CA_11D2_A30A_00105A60F930__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// WebView.h : header file
//

//      Quarterview
//      joseph j. kim, march 1999
//      copyright 1999      quarterview

////////////////////////////////////
// CWebView html view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif
#include <afxhtml.h>

class CWebView : public CHtmlView
{
public:
    CWebView();          // protected constructor used by dynamic creation

protected:
    DECLARE_DYNCREATE(CWebView)

// html Data
public:
    //{AFX_DATA(CWebView)
    // NOTE: the ClassWizard will add data members here
    //}AFX_DATA

// Attributes
public:

// Operations
public:
    void OnNavigateBack();
    void OnNavigateForward();
    void OnNavigateHome();
    void OnNavigateStop();
    void OnGoBannerAd();

    void GoBannerAd() { OnGoBannerAd(); }

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CWebView)
public:
    virtual void OnInitialUpdate();
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
}}AFX_VIRTUAL

};

#endif
```

WebView.h

```
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
//}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CWebView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
    //{{AFX_MSG(CWebView)
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef(AFX_CWEBVIEW_H__EC3381DA_D9CA_11D2_A30A_00105A60F930__INCLUDED_)
```

WhisperDlg.cpp

```
// WhisperDlg.cpp : implementation file
//

#include "stdafx.h"
#include "UC2.h"
#include "WhisperDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////

// CWhisperDlg dialog

CWhisperDlg::CWhisperDlg(CWnd* pParent /*=NULL*/)
: CDialog(CWhisperDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CWhisperDlg)
    m_strText = _T("");
    m_strToID = _T("");
    //}}AFX_DATA_INIT
}

void CWhisperDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CWhisperDlg)
    DDX_Text(pDX, IDC_EDIT_TEXT, m_strText);
    DDX_Text(pDX, IDC_STATIC_TO, m_strToID);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CWhisperDlg, CDialog)
    //{{AFX_MSG_MAP(CWhisperDlg)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CWhisperDlg message handlers
```


WhisperDlg.h

```

#if
!defined(AFX_WHISPERDLG_H__4E6B0EC2_C674_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_WHISPERDLG_H__4E6B0EC2_C674_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// WhisperDlg.h : header file
//

////////////////////

// CWhisperDlg dialog

class CWhisperDlg : public CDialog
{
// Construction
public:
    CWhisperDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CWhisperDlg)
    enum { IDD = IDD_DIALOG_WHISPER };
    CString        m_strText;
    CString        m_strToID;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWhisperDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CWhisperDlg)
    // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_WHISPERDLG_H__4E6B0EC2_C674_11D1_80E2_080009B9F339__INCLUDED_)

```

```

// CloseDlg.cpp : implementation file
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT.COM
//=====

#include "stdafx.h"
#include "MapEd.h"
#include "CloseDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CClosedlg dialog

CClosedlg::CClosedlg(CWnd* pParent /*=NULL*/)
    : CDialog(CClosedlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CClosedlg)
    // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}

void CClosedlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CClosedlg)
    // NOTE: the ClassWizard will add DDX and DDV calls here
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CClosedlg, CDialog)
   //{{AFX_MSG_MAP(CClosedlg)
    // NOTE: the ClassWizard will add message map macros here
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CClosedlg message handlers

```

```

#if !defined(AFX_CLOSEDLG_H_D5010EC7_A1F9_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_CLOSEDLG_H_D5010EC7_A1F9_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// CloseDlg.h : header file
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

////////////////////
// CClosedDlg dialog

class CClosedDlg : public CDialog
{
// Construction
public:
    CClosedDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CClosedDlg)
    enum { IDD = IDD_CLOSE };
        // NOTE: the ClassWizard will add data members here
    }}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CClosedDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CClosedDlg)
        // NOTE: the ClassWizard will add member functions here
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_CLOSEDLG_H_D5010EC7_A1F9_11D1_80E2_080009B9F339__INCLUDED_)

```

```

// ElevDlg.cpp : implementation file
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "MapEd.h"
#include "ElevDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CElevDlg dialog

CElevDlg::CElevDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CElevDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CElevDlg)
    m_nElevNew = 0;
    m_strElevCur = _T("");
    //}}AFX_DATA_INIT
}

void CElevDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CElevDlg)
    DDX_Text(pDX, IDC_EDIT_NEW_ELEV, m_nElevNew);
    DDV_MinMaxInt(pDX, m_nElevNew, 0, 300);
    DDX_Text(pDX, IDC_ST_CURRENT_ELEV, m_strElevCur);
    DDV_MaxChars(pDX, m_strElevCur, 5);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CElevDlg, CDialog)
    //{{AFX_MSG_MAP(CElevDlg)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CElevDlg message handlers

```

```

#if !defined(AFX_ELEVDLG_H__857FFE42_A782_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_ELEVDLG_H__857FFE42_A782_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ElevDlg.h : header file
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT NETWORKS INC
//=====

////////////////////
// CElevDlg dialog

class CElevDlg : public CDialog
{
// Construction
public:
    CElevDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CElevDlg)
    enum { IDD = IDD_ELEVATION };
    int         m_nElevNew;
    CString     m_strElevCur;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CElevDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CElevDlg)
    // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_ELEVDLG_H__857FFE42_A782_11D1_80E2_080009B9F339__INCLUDED_)

```

```

// GetIntDlg.cpp : implementation file
//

#include "stdafx.h"
#include "MapEd.h"
#include "GetIntDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CGetIntDlg dialog

CGetIntDlg::CGetIntDlg(CWnd* pParent /*=NULL*/)
: CDialog(CGetIntDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CGetIntDlg)
    m_nNew = 0;
    m_strCur = _T("");
    //}}AFX_DATA_INIT
    m_strCaption.Empty();
}

void CGetIntDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CGetIntDlg)
    DDX_Text(pDX, IDC_EDIT_NEW_INT, m_nNew);
    DDX_Text(pDX, IDC_ST_CURRENT_INT, m_strCur);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGetIntDlg, CDialog)
    //{{AFX_MSG_MAP(CGetIntDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// CGetIntDlg message handlers

BOOL CGetIntDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    if (!m_strCaption.IsEmpty())
        SetWindowText(m_strCaption);

    m_strCur.Format("%d", m_nNew);    // So, set m_nNew before calling
DoModal
    UpdateData(FALSE);    // Initialize Dialog
}

```

MapEd\GetIntDlg.cpp

```
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
```

```

#if
!defined(AFX_GETINTDLG_H__4F156CC2_AA1E_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_GETINTDLG_H__4F156CC2_AA1E_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// GetIntDlg.h : header file
//

////////////////////

// CGetIntDlg dialog

class CGetIntDlg : public CDialog
{
// Construction
public:
    CString m_strCaption;
    CGetIntDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CGetIntDlg)
    enum { IDD = IDD_GETINT };
    int         m_nNew;
    CString     m_strCur;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CGetIntDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CGetIntDlg)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_GETINTDLG_H__4F156CC2_AA1E_11D1_80E2_080009B9F339__INCLUDED_)

```



```
// GetTextDlg.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "MapEd.h"
#include "GetTextDlg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CGetTextDlg dialog
```

```
CGetTextDlg::CGetTextDlg(CWnd* pParent /*=NULL*/)
: CDialog(CGetTextDlg::IDD, pParent)
```

```
{
   //{{AFX_DATA_INIT(CGetTextDlg)
    m_strText = _T("");
    //}}AFX_DATA_INIT
    m_strCaption.Empty();
}
```

```
void CGetTextDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CGetTextDlg)
    DDX_Text(pDX, IDC_EDIT_STRING, m_strText);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CGetTextDlg, CDialog)
```

```
    //{{AFX_MSG_MAP(CGetTextDlg)
    //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CGetTextDlg message handlers
```

```
BOOL CGetTextDlg::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();

    if (!m_strCaption.IsEmpty())
        SetWindowText(m_strCaption);

    UpdateData(FALSE); // Initialize Dialog
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
```

```

#if
!defined(AFX_GETTEXTDLG_H__C58C7FE4_B59C_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_GETTEXTDLG_H__C58C7FE4_B59C_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// GetTextDlg.h : header file
//

/////////////////////////////////////////////////////////////////
// CGetTextDlg dialog

class CGetTextDlg : public CDialog
{
// Construction
public:
    CString m_strCaption;
    CGetTextDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CGetTextDlg)
    enum { IDD = IDD_GETSTRING };
    CString    m_strText;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CGetTextDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CGetTextDlg)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_GETTEXTDLG_H__C58C7FE4_B59C_11D1_80E2_080009B9F339__INCLUDED_)

```

MapEd\MainFrm.cpp

// MainFrm.cpp : implementation of the CMainFrame class

//

//=====

// (C) Programmed by Soomin Kim, Feb 1998

// Information Technology Institute

// UNICHAT NETWORKS INC

//=====

#include "stdafx.h"

#include "MapEd.h"

#include "MainFrm.h"

#include "MapEdView.h"

#include "MapListView.h"

#include "MapEdDoc.h"

#include "CloseDlg.h"

#ifdef _DEBUG

#define new DEBUG_NEW

#undef THIS_FILE

static char THIS_FILE[] = __FILE__;

#endif

////////////////////////////////////

// CMainFrame

```
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
```

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
```

```
    ON_WM_QUERYNEWPALETTE()
```

```
    ON_WM_PALETTECHANGED()
```

```
    //{{AFX_MSG_MAP(CMainFrame)
```

```
    ON_WM_CREATE()
```

```
    ON_WM_CLOSE()
```

```
    //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
static UINT indicators[] =
```

```
{
```

```
    ID_SEPARATOR,           // status line indicator
```

```
    ID_INDICATOR_CAPS,
```

```
    ID_INDICATOR_NUM,
```

```
    ID_INDICATOR_SCRL,
```

```
};
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
// CMainFrame construction/destruction
```

```
CMainFrame::CMainFrame()
```

```
{
```

```
    TRACE0("CMainFrame::CMainFrame()\n");
```

```
}
```

```
CMainFrame::~CMainFrame()
```

```

{
    TRACE0("CMainFrame::~~CMainFrame()\n");
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    // CG: This line was added by the Palette Support component
    m_pPalette = NULL;

    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    m_wndToolBar.SetFlatLookStyle();    // CFlatToolBar

```

```

// TODO: Remove this if you don't want tool tips or a resizable
toolbar

m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

// TODO: Delete these three lines if you don't want the toolbar to
// be dockable

m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);

return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    /*
    // Create a window without min/max buttons or sizable border
    cs.style = WS_OVERLAPPED | WS_SYSMENU | WS_BORDER;

    // Size the window and center it
    cs.cy = ::GetSystemMetrics(SM_CYSCREEN) - 200;
    cs.cx = ::GetSystemMetrics(SM_CXSCREEN) - 200;
    cs.y = ((cs.cy + 200) - cs.cy) / 2;
    cs.x = ((cs.cx + 200) - cs.cx) / 2;
    */

    return CFrameWnd::PreCreateWindow(cs);
}

```

```
////////////////////////////////////
```

```
// CMainFrame diagnostics
```

```
#ifdef _DEBUG
```

```
void CMainFrame::AssertValid() const
```

```
{
```

```
    CFrameWnd::AssertValid();
```

```
}
```

```
void CMainFrame::Dump(CDumpContext& dc) const
```

```
{
```

```
    CFrameWnd::Dump(dc);
```

```
}
```

```
#endif //_DEBUG
```

```
////////////////////////////////////
```

```
// CMainFrame message handlers
```

```
BOOL CMainFrame::OnQueryNewPalette()
```

```
{
```

```
    CMapEdDoc* pDoc = (CMapEdDoc*)GetActiveDocument();
```

```
    CMapEdView* pMEV = pDoc->GetMapEdView();
```

```
    CMapListView* pMLV = pDoc->GetMapListView();
```

```
    if (pMEV)
```

```
        pMEV->SendMessage(WM_QUERYNEWPALETTE, (WPARAM)0, (LPARAM)0);
```

```
    if (pMLV)
```

```
        pMLV->SendMessage(WM_QUERYNEWPALETTE, (WPARAM)0, (LPARAM)0);
```

```
    // CG: This function was added by the Palette Support component
```

```

    if (m_pPalette == NULL)
        return FALSE;

    // BLOCK
    {
        CClientDC dc(this);
        CPalette* pOldPalette = dc.SelectPalette(m_pPalette,
            GetCurrentMessage()->message == WM_PALETTECHANGED);
        UINT nChanged = dc.RealizePalette();
        dc.SelectPalette(pOldPalette, TRUE);

        if (nChanged == 0)
            return FALSE;
    }

    Invalidate();

    return TRUE;
}

void CMainFrame::OnPaletteChanged(CWnd* pFocusWnd)
{
    // CG: This function was added by the Palette Support component
    if (pFocusWnd == this || IsChild(pFocusWnd))
        return;

    // Pass this message on to the active view (OSBVeiw-derived)
    CMapEdDoc* pDoc = (CMapEdDoc*)GetActiveDocument();
    CMapEdView* pMEV = pDoc->GetMapEdView();
    CMapListView* pMLV = pDoc->GetMapListView();

    if (pMEV)
        pMEV->SendMessage(WM_PALETTECHANGED,
            (LPARAM) (pFocusWnd->GetSafeHwnd()),
            (LPARAM) 0);

    if (pMLV)
        // OnPaletteChanged is not public

```



```

        pMLV->SendMessage(WM_PALETTECHANGED,

                           (WPARAM) (pFocusWnd->GetSafeHwnd()),

                           (LPARAM) 0);

    OnQueryNewPalette();
}

CPalette* CMainFrame::SetPalette(CPalette* pPalette)
{
    // CG: This function was added by the Palette Support component

    // Call this function when the palette changes. It will
    // realize the palette in the foreground to cause the screen
    // to repaint correctly. All calls to CDC::SelectPalette in
    // painting code should select palettes in the background.

    CPalette* pOldPalette = m_pPalette;
    m_pPalette = pPalette;
    OnQueryNewPalette();
    return pOldPalette;
}

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext)
{
    // create a splitter with 1 row, 2 columns
    if (!m_wndSplitter.CreateStatic(this, 1, 2))
    {
        TRACE0("Failed to create static splitters\n");

        return FALSE;    // failed to create
    }

    // add the first splitter pane - the default view in column 0
    if (!m_wndSplitter.CreateView(0, 0,

        RUNTIME_CLASS(CMapEdView), CSize(590, 50), pContext))
    {
        TRACE0("Failed to create first pane\n");
    }
}

```

```
        return FALSE;
    }

    // add the second splitter pane - an input view in column 1
    if (!m_wndSplitter.CreateView(0, 1,
        RUNTIME_CLASS(CMapListView), CSize(0, 0), pContext))
    {
        TRACE0("Failed to create second pane\n");
        return FALSE;
    }

    // activate the input view
    SetActiveView((CView*)m_wndSplitter.GetPane(0,1));

    return TRUE;
}

void CMainFrame::OnClose()
{
    CCloseDlg dlg;
    if (dlg.DoModal() == IDCANCEL)
        return;

    CFrameWnd::OnClose();
}
```

```

// MainFrm.h : interface of the CMainFrame class
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC.
//=====

#ifdef !defined(AFX_MAINFRM_H__7ECC1E0A_9C09_11D1_80E2_288A06C10000__INCLUDED_)
#define AFX_MAINFRM_H__7ECC1E0A_9C09_11D1_80E2_288A06C10000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "../FlatToolBar.h"

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:
    void SetStatusBarText(const char* msg) {
m_wndStatusBar.SetWindowText(msg); }
    CPalette* SetPalette(CPalette* pPalette);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar          m_wndStatusBar;
    CFlatToolBar        m_wndToolBar;

// Generated message map functions
protected:
    CSplitterWnd m_wndSplitter;
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
    afx_msg BOOL OnQueryNewPalette();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);

```

```

CPalette* m_pPalette;
//{{AFX_MSG(CMainFrame)
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
afx_msg void OnClose();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_MAINFRM_H__7ECC1E0A_9C09_11D1_80E2_288A06C10000__INCLUDED_)

```

```

// MapEd.cpp : Defines the class behaviors for the application.
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "MapEd.h"

#include "MainFrm.h"
#include "MapEdDoc.h"
#include "MapEdView.h"

#include "../Stage.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CMapEdApp

BEGIN_MESSAGE_MAP(CMapEdApp, CWinApp)
   //{{AFX_MSG_MAP(CMapEdApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // NOTE - the ClassWizard will add and remove mapping macros
    here.
    //      DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

////////////////////////////////////
// CMapEdApp construction

CMapEdApp::CMapEdApp()
{
    m_pStage = NULL;
    m_bPause = false;
}

////////////////////////////////////
// The one and only CMapEdApp object

CMapEdApp theApp;

////////////////////////////////////
// CMapEdApp initialization

```

```

BOOL CMapEdApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC
statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings(); // Load standard INI file options (including
MRU)

    // Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CMapEdDoc),
        RUNTIME_CLASS(CMainFrame),           // main SDI frame window
        RUNTIME_CLASS(CMapEdView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:

```

```

    CAboutDlg();

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
    // No message handlers
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
    }}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
    }}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    }}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CMapEdApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMapEdApp commands

BOOL CMapEdApp::OnIdle(LONG lCount)
{
    // CG: The following code inserted by 'Idle Time Processing' component.

    // Note: Do not perform lengthy tasks during OnIdle because your
    // application cannot process user input until OnIdle returns.

```

```

        // call the base class
        CWinApp::OnIdle(lCount);

        if (m_bPause)
            return FALSE;

/*
    if (lCount == 0)
    {
        // TODO: add code to perform important idle time processing
    }
    else if (lCount == 100)
    {
        // TODO: add code to perform less important tasks during idle
    }
    else if (lCount == 1000)
    {
    }
*/
    // This is the Heart of the Program!
    if (m_pStage)
    {
        m_pStage->TickAll();
        return TRUE;
    }

    // return FALSE when there is no more idle processing to do
    return FALSE;
}

BOOL CMapEdApp::PreTranslateMessage(MSG* pMsg)
{
    return CWinApp::PreTranslateMessage(pMsg);
}

```



```

// MapEd.h : main header file for the MAPED application
//
//=====
//      (C) Programmed by Soomin Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#ifdef !defined(AFX_MAPED_H__7ECC1E06_9C09_11D1_80E2_288A06C10000__INCLUDED_)
#define AFX_MAPED_H__7ECC1E06_9C09_11D1_80E2_288A06C10000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CMapEdApp:
// See MapEd.cpp for the implementation of this class
//

class CStage;

class CMapEdApp : public CWinApp
{
public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    virtual BOOL OnIdle(LONG lCount);
    CMapEdApp();

    void SetStage(CStage* pStage)    { m_pStage = pStage; }
    void SetPause(const BOOL bPause) { m_bPause = bPause; }

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CMapEdApp)
public:
    virtual BOOL InitInstance();
//}}AFX_VIRTUAL

// Implementation
protected:
    CStage*          m_pStage;
    BOOL             m_bPause;

    //{{AFX_MSG(CMapEdApp)
    afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //      DO NOT EDIT what you see in these blocks of generated code
    !
    //}}AFX_MSG

```

```
        DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_MAPED_H__7ECC1E06_9C09_11D1_80E2_288A06C10000__INCLUDED_)
```

```

// MapEdDoc.cpp : implementation of the CMapEdDoc class
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "MapEd.h"

#include "MapEdDoc.h"
#include "MapEdView.h"
#include "MapListView.h"
#include "../Stage.h"
#include "../TileMap.h"
#include "PalDlg.h"
#include "MapEnvDlg.h"
#include "GetTextDlg.h"
#include "GetIntDlg.h"

#include "../UC2Ani/DIB.h"
#include "../UC2Ani/PhSprite.h"

#include "../ResMan.h"
#include "../Parser.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CParser      gParser(1024*4); // to be used as an extern variable in other
files
CResMan      gResMan;

double GetMapEditorVersion();

////////////////////////////////////
// CMapEdDoc

IMPLEMENT_DYNCREATE(CMapEdDoc, CDocument)

BEGIN_MESSAGE_MAP(CMapEdDoc, CDocument)
    //{{AFX_MSG_MAP(CMapEdDoc)
    ON_COMMAND(ID_VIEW_GRID, OnViewGrid)
    ON_UPDATE_COMMAND_UI(ID_VIEW_GRID, OnUpdateViewGrid)
    ON_COMMAND(ID_VIEW_EARTH, OnViewEarth)
    ON_UPDATE_COMMAND_UI(ID_VIEW_EARTH, OnUpdateViewEarth)
    ON_COMMAND(ID_VIEW_PALETTE, OnViewPalette)
    ON_COMMAND(ID_OPTIONS_ENV, OnOptionsEnv)
    ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
    ON_COMMAND(ID_FILE_SAVE, OnFileSave)
    ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
    ON_COMMAND(ID_EDIT_MODE, OnEditMode)
    ON_UPDATE_COMMAND_UI(ID_EDIT_MODE, OnUpdateEditMode)
    //}}

```

```

ON_COMMAND(ID_VIEW_MAP_ONLY, OnViewMapOnly)
ON_UPDATE_COMMAND_UI(ID_VIEW_MAP_ONLY, OnUpdateViewMapOnly)
ON_COMMAND(ID_VIEW_GRAPH, OnViewGraph)
ON_UPDATE_COMMAND_UI(ID_VIEW_GRAPH, OnUpdateViewGraph)
ON_COMMAND(ID_EDIT_TILE_COORD, OnEditTileCoord)
ON_UPDATE_COMMAND_UI(ID_EDIT_TILE_COORD, OnUpdateEditTileCoord)
ON_COMMAND(ID_VIEW_PAUSE, OnViewPause)
ON_UPDATE_COMMAND_UI(ID_VIEW_PAUSE, OnUpdateViewPause)
ON_COMMAND(ID_MENU_STAGE_TITLE, OnMenuStageTitle)
ON_COMMAND(ID_EDIT_STAND, OnEditStand)
ON_UPDATE_COMMAND_UI(ID_EDIT_STAND, OnUpdateEditStand)
ON_COMMAND(ID_EDIT_CREATE_MODE, OnEditCreateMode)
ON_UPDATE_COMMAND_UI(ID_EDIT_CREATE_MODE, OnUpdateEditCreateMode)
ON_COMMAND(ID_MENU_RECALC_DA, OnMenuRecalcDa)
ON_COMMAND(ID_MENU_SYNC_EA, OnMenuSyncEa)
ON_COMMAND(ID_VIEW_NOELEV, OnViewNoelev)
ON_UPDATE_COMMAND_UI(ID_VIEW_NOELEV, OnUpdateViewNoelev)
ON_COMMAND(ID_MENU_STAGE_MUSIC, OnMenuStageMusic)
ON_COMMAND(ID_VIEW_EA_GRAPH, OnViewEaGraph)
ON_UPDATE_COMMAND_UI(ID_VIEW_EA_GRAPH, OnUpdateViewEaGraph)
ON_COMMAND(ID_MENU_ELEVATION, OnMenuElevation)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CMapEdDoc construction/destruction

CMapEdDoc::CMapEdDoc()
{
    TRACE0("CMapEdDoc::CMapEdDoc()\n");
    m_pStage = NULL;
    m_bViewEarth = FALSE;
    m_bViewGraph = FALSE;
    m_bViewEAGraph = FALSE;
    m_nEditMode = SPRITE_EDIT_MODE;
    m_bCreateMode = TRUE;
    m_bTileCoord = TRUE;
    m_bPause = FALSE;
    m_bStandEdit = FALSE;
}

CMapEdDoc::~CMapEdDoc()
{
    TRACE0("CMapEdDoc::~CMapEdDoc()\n");
    if (m_pStage)
    {
        EndAnimation();
        delete m_pStage;
    }
}

BOOL CMapEdDoc::OnNewDocument()
{
    TRACE0("CMapEdDoc::OnNewDocument()\n");
    EndAnimation();
    if (!CDocument::OnNewDocument())
        return FALSE;
}

```

```

CMapView* pView = GetMapView();
ASSERT(pView);

if (m_pStage)
{
    m_pStage->DeleteStage();
    m_pStage->ClearFilename();
}
else
{
    m_pStage = new CStage;
    m_pStage->Initialize(pView);
}

CMapEnvDlg dlg;
dlg.m_strDataPath = *gResMan.GetResPath();
dlg.m_strPalFile = *m_pStage->GetPalFileName();
CTileMap* pTM = m_pStage->GetTileMap();
CSize szT = pTM ? pTM->GetTileSize() : CSize(64, 32);
CSize szScr = pTM ? pTM->GetScreenSize() : CSize(640, 368); //640,
384);
dlg.m_nTileWidth = szT.cx;
dlg.m_nTileHeight = szT.cy;
dlg.m_nScrWidth = szScr.cx;
dlg.m_nScrHeight = szScr.cy;
if (pTM)
{
    if (AfxMessageBox("Delete current Stage?") == IDCANCEL)
        return FALSE;
    m_pStage->ClearFilename();
    m_pStage->DeleteStage();
}
if (dlg.DoModal() == IDOK)
{
    szT = CSize(dlg.m_nTileWidth, dlg.m_nTileHeight);
    szScr = CSize(dlg.m_nScrWidth, dlg.m_nScrHeight);
    gResMan.SetResPath(dlg.m_strDataPath);
    m_pStage->SetPalFileName(dlg.m_strPalFile);
}

if (!m_pStage->CreateStage(szT, szScr))
    return FALSE;
pView->AdjustFrameWindow();
BeginAnimation();

UpdateAllViews(NULL); // COSBView::OnUpdate() - Render & Draw
// CMapView::OnUpdate -
UpdateList()
// SetModifiedFlag(FALSE);
return TRUE;
}

void CMapEdDoc::DeleteContents()
{
    TRACE("CMapEdDoc::DeleteContents()\n");
    SetModifiedFlag(FALSE);
}

```

```

        CDocument::DeleteContents();
    }

    //////////////////////////////////////////////////
    // CMapEdDoc serialization

void CMapEdDoc::Serialize(CArchive& ar)
{
    CDocument::Serialize(ar);
    if (ar.IsStoring())
    {
        /*
            m_SpriteList.Serialize(ar);
        */
    }
    else // Reading
    {
        /*
            // read the sprite list
            CMapEdView* pView = GetMapEdView();
            ASSERT(pView);
            m_SpriteList.m_NotifyObj.SetView(pView);
            m_SpriteList.Serialize(ar);

            SetModifiedFlag(FALSE);
            UpdateAllViews(NULL, 0, NULL);
        */
    }
}

    //////////////////////////////////////////////////
    // CMapEdDoc diagnostics

#ifdef _DEBUG
void CMapEdDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CMapEdDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

    //////////////////////////////////////////////////
    // CMapEdDoc commands

CMapEdView* CMapEdDoc::GetMapEdView()
{
    POSITION pos = GetFirstViewPosition();
    ASSERT(pos);
    CMapEdView* pView = (CMapEdView*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CMapEdView)));
    return pView;
}

```

```

CMapView* CMapEdDoc::GetMapView()
{
    POSITION pos = GetFirstViewPosition();
    ASSERT(pos);
    GetNextView(pos);
    CMapView* pView = (CMapView*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CMapView)));
    return pView;
}

void CMapEdDoc::SetTitle(LPCTSTR lpszTitle)
{
    // UNREFERENCED_PARAMETER(lpszTitle);

    CString strTitle;
    strTitle.Format("Turtle Quarter View Editor V%.2f ",
GetMapEditorVersion());
    if (lpszTitle)
        strTitle += lpszTitle;
    CDocument::SetTitle(strTitle);
}

void CMapEdDoc::OnOptionsEnv()
{
    if (!m_pStage)
        return;
    CMapEnvDlg dlg;
    dlg.m_strDataPath = *gResMan.GetResPath();
    dlg.m_strPalFile = *m_pStage->GetPalFileName();
    CTileMap* pTM = m_pStage->GetTileMap();
    if (!pTM)
        return;
    CSize szT(pTM->GetTileSize());
    CSize szScr(pTM->GetScreenSize());
    dlg.m_nTileWidth = szT.cx;
    dlg.m_nTileHeight = szT.cy;
    dlg.m_nScrWidth = szScr.cx;
    dlg.m_nScrHeight = szScr.cy;
    dlg.SetReadOnly();
    if (dlg.DoModal() == IDOK)
    {
        gResMan.SetResPath(dlg.m_strDataPath);
        m_pStage->SetPalFileName(dlg.m_strPalFile);
    }
}

void CMapEdDoc::OnFileOpen()
{
    if (!m_pStage)
        return;
    EndAnimation();
    m_pStage->Load();
    CMapView* pView = GetMapView();
    pView->AdjustFrameWindow();
    BeginAnimation();
}

```

```

        UpdateAllViews(NULL);
        UpdateTitle();
    }

void CMapEdDoc::OnFileSave()
{
    if (!m_pStage)
        return;
    m_pStage->Save(*m_pStage->GetFileName());
    CMapListView* pView = GetMapListView();
    UpdateAllViews(pView);
    UpdateTitle();
}

void CMapEdDoc::OnFileSaveAs()
{
    if (!m_pStage)
        return;
    m_pStage->Save();
    CMapListView* pView = GetMapListView();
    UpdateAllViews(pView);
    UpdateTitle();
}

void CMapEdDoc::UpdateTitle()
{
    CString strTitle;
    CString* pS1 = m_pStage->GetFileName();
    CString* pS2 = m_pStage->GetMapFileName();
    CString* pS3 = m_pStage->GetPalFileName();
    if (pS1)
    {
        strTitle = "[" + *pS1 + "]";
    }
    if (pS2)
    {
        strTitle += "[";
        strTitle += *pS2;
        strTitle += "]";
    }
    if (pS3)
    {
        strTitle += "[";
        strTitle += *pS3;
        strTitle += "]";
    }
    SetTitle(strTitle);
}

////////////////////////////////////
// Edit Mode

void CMapEdDoc::OnEditMode()
{
    m_nEditMode = (m_nEditMode == TILE_EDIT_MODE) ?
        SPRITE_EDIT_MODE :
        TILE_EDIT_MODE;
}

```



```

}

void CMapEdDoc::OnUpdateEditMode(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_nEditMode == TILE_EDIT_MODE);
}

void CMapEdDoc::OnEditTileCoord()
{
    m_bTileCoord = !m_bTileCoord;
}

void CMapEdDoc::OnUpdateEditTileCoord(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bTileCoord);
}

void CMapEdDoc::OnViewNoelev() // No Elevation
{
    if (!m_pStage)
        return;
    WORD wRM = m_pStage->GetRenderMode();
    wRM ^= RM_ELEV;
    m_pStage->SetRenderMode(wRM);
    m_bPause = ((wRM & RM_ELEV) != RM_ELEV);
    PauseAnimation();
    CMapListView* pView = GetMapListView();
    UpdateAllViews(pView);
}

void CMapEdDoc::OnUpdateViewNoelev(CCmdUI* pCmdUI)
{
    if (!m_pStage)
        return;
    WORD wRM = m_pStage->GetRenderMode();
    pCmdUI->SetCheck((wRM & RM_ELEV) == RM_ELEV);
}

void CMapEdDoc::OnViewMapOnly()
{
    if (!m_pStage)
        return;
    WORD wRM = m_pStage->GetRenderMode();
    wRM ^= RM_SPRITE;
    m_pStage->SetRenderMode(wRM);

    m_nEditMode = ((wRM & RM_SPRITE) == RM_SPRITE) ? SPRITE_EDIT_MODE :
TILE_EDIT_MODE;
    CMapListView* pView = GetMapListView();
    UpdateAllViews(pView);
}

void CMapEdDoc::OnUpdateViewMapOnly(CCmdUI* pCmdUI)
{
    if (!m_pStage)
        return;
    WORD wRM = m_pStage->GetRenderMode();

```

```

        pCmdUI->Enable((wRM & RM_ELEV) != RM_ELEV);        // enable only for no
elevation mode
        pCmdUI->SetCheck((wRM & RM_SPRITE) != RM_SPRITE);
    }

void CMapEdDoc::OnViewGraph()
{
    m_bViewGraph = !m_bViewGraph;
    m_bPause = m_bViewGraph;
    PauseAnimation();
    CMapEdView* pView = GetMapEdView();
    if (pView)
        pView->Invalidate(FALSE);
}

void CMapEdDoc::OnUpdateViewGraph(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bViewGraph);
}

void CMapEdDoc::OnViewEaGraph()
{
    m_bViewEAGraph = !m_bViewEAGraph;
    m_bPause = m_bViewEAGraph;
    PauseAnimation();
    CMapEdView* pView = GetMapEdView();
    if (pView)
        pView->Invalidate(FALSE);
}

void CMapEdDoc::OnUpdateViewEaGraph(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bViewEAGraph);
}

void CMapEdDoc::OnViewGrid()
{
    if (!m_pStage)
        return;
    WORD wRM = m_pStage->GetRenderMode();
    wRM ^= RM_GRID;
    m_pStage->SetRenderMode(wRM);
    CMapListView* pView = GetMapListView();
    UpdateAllViews(pView);
}

void CMapEdDoc::OnUpdateViewGrid(CCmdUI* pCmdUI)
{
    if (!m_pStage)
        return;
    WORD wRM = m_pStage->GetRenderMode();
    pCmdUI->SetCheck((wRM & RM_GRID) == RM_GRID);
}

void CMapEdDoc::OnViewEarth()
{
}

```

```

void CMapEdDoc::OnUpdateViewEarth(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bViewEarth);
}

void CMapEdDoc::OnViewPalette()
{
    CPalDlg          palDlg;

    CMapEdView* pView = GetMapEdView();
    if (pView)
        palDlg.SetPalette(pView->GetOSBPalette());
    palDlg.DoModal();
}

void CMapEdDoc::OnViewPause()
{
    m_bPause = !m_bPause;
    PauseAnimation();
}

void CMapEdDoc::OnUpdateViewPause(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bPause);
}

void CMapEdDoc::OnMenuStageTitle()
{
    if (!m_pStage)
        return;
    CGetTextDlg dlg;
    dlg.m_strCaption.LoadString(IDS_ENTER_STAGE_TITLE);
    CString strTitle(*m_pStage->GetTitle());
    CString strFile(*m_pStage->GetFileName());
    gResMan.MakeResName(strFile);
    if (strTitle.IsEmpty())
    {
        strTitle.Format("[u2/%s]", strFile);
    }
    else
    {
        int i = strTitle.Find('/');
        if (i <= 0)
        {
            if (strTitle[0] == '[') // [U2] or [u2]
            {
                CString strTemp;
                int nCount = strTitle.GetLength();
                if (nCount >= 4)
                {
                    strTemp = strTitle.Right(nCount-4);
                    strTemp.TrimLeft();
                    strTitle.Format("[u2/%s]", strFile);
                    strTitle += strTemp;
                }
            }
        }
    }
}

```

```

    }
}
dlg.m_strText = strTitle;
if (dlg.DoModal() == IDOK)
    m_pStage->SetTitle(dlg.m_strText);
}

void CMapEdDoc::OnMenuStageMusic()
{
    if (!m_pStage)
        return;
    CGetTextDlg dlg;
    dlg.m_strCaption.LoadString(IDS_ENTER_STAGE_MUSIC);
    dlg.m_strText = *m_pStage->GetMusicSeq(); // "[U2]"
    if (dlg.DoModal() == IDOK)
        m_pStage->SetMusicSeq(dlg.m_strText);
}

void CMapEdDoc::OnEditStand()
{
    m_bStandEdit = !m_bStandEdit;
    if (!m_pStage)
        return;

    WORD wRM = m_pStage->GetRenderMode();
    if (m_bStandEdit)
        wRM |= RM_GRID;
    else
        wRM &= ~RM_GRID;
    m_pStage->SetRenderMode(wRM);
    CMapListView* pView = GetMapListView();
    UpdateAllViews(pView);
}

void CMapEdDoc::OnUpdateEditStand(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bStandEdit);
}

void CMapEdDoc::OnEditCreateMode()
{
    m_bCreateMode = !m_bCreateMode;
}

void CMapEdDoc::OnUpdateEditCreateMode(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_bCreateMode);
}

void CMapEdDoc::OnMenuRecalcDa()
{
    CMapEdView* pView = GetMapEdView();
    pView->RecalculateDA();
    CMapListView* pMLV = GetMapListView();
    UpdateAllViews(pMLV);
    AfxMessageBox("Recalculating DA(Direction Attribute) - Done");
}

```

```

void CMapEdDoc::OnMenuSyncEa()
{
    CMapEdView* pView = GetMapEdView();
    pView->SynchronizeEA();
    CMapListView* pMLV = GetMapListView();
    UpdateAllViews(pMLV);
    AfxMessageBox("Synchronizing EA(Elevation for Actor) with sprite
elevation - Done");
}

void CMapEdDoc::OnMenuElevation()
{
    CGetIntDlg dlg;
    dlg.m_strCaption.LoadString(IDS_ENTER_TILE_ELEVS);
    dlg.m_nNew = 0;
    if (dlg.DoModal() != IDOK)
        return;

    CTileMap* pTM = m_pStage->GetTileMap();
    pTM->IncreaseElevations(dlg.m_nNew * ELEVATION_UNIT);

    CMapListView* pView = GetMapListView();
    UpdateAllViews(pView);
}

```

```

// MapEdDoc.h : interface of the CMapEdDoc class
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#ifdef !defined(AFX_MAPEDDOC_H__7ECC1E0C_9C09_11D1_80E2_288A06C10000__INCLUDED_)
#define AFX_MAPEDDOC_H__7ECC1E0C_9C09_11D1_80E2_288A06C10000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CDIB;
class CMapEdView;
class CMapListView;
class CResMan;
class CTiles;
class CStage;

enum MAP_EDIT_MODE
{
    TILE_EDIT_MODE,
    SPRITE_EDIT_MODE,
    STAND_EDIT_MODE
};

class CMapEdDoc : public CDocument
{
protected: // create from serialization only
    CMapEdDoc();
    DECLARE_DYNCREATE(CMapEdDoc)

// Attributes
public:
    CMapEdView*      GetMapEdView();
    CMapListView*    GetMapListView();
    CStage*          GetStage()          { return m_pStage; }
    void             UpdateTitle();
    BOOL             IsTileEditMode() const
                    { return (m_nEditMode == TILE_EDIT_MODE); }
    BOOL             IsStandEditMode() const
                    { return m_bStandEdit; }
    BOOL             IsViewGraph() const { return m_bViewGraph; }
    BOOL             IsViewEAGraph() const { return m_bViewEAGraph; }
    BOOL             IsTileCoordMode() const { return m_bTileCoord; }
    BOOL             IsCreateMode() const { return m_bCreateMode; }

// Operations
public:
    void             SetEditMode(const int nMode) { m_nEditMode = nMode; }
}
    void             SetTileCoordMode(const BOOL bTC) { m_bTileCoord =
bTC; }
    void             BeginAnimation() const

```

```

        { ((CMapEdApp*)AfxGetApp()) -
>SetStage(m_pStage); }
    void                EndAnimation() const
                        { ((CMapEdApp*)AfxGetApp())->SetStage(NULL); }
    void                PauseAnimation() const
                        { ((CMapEdApp*)AfxGetApp()) -
>SetPause(m_bPause); }

private:
    BOOL                m_bViewEarth;
    BOOL                m_bViewGraph;
    BOOL                m_bViewEAGraph;
    int                 m_nEditMode;        // 0: Tile Edit, 1: Sprite
Edit
    BOOL                m_bStandEdit;
    BOOL                m_bCreateMode;      // LBUTTONDONW => Create sprite
    BOOL                m_bTileCoord;
    BOOL                m_bPause;          // Pause Animation
    CStage*             m_pStage;

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CMapEdDoc)
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual void SetTitle(LPCTSTR lpszTitle);
    virtual void DeleteContents();
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMapEdDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:

// Generated message map functions
protected:
   //{{AFX_MSG(CMapEdDoc)
    afx_msg void OnViewGrid();
    afx_msg void OnUpdateViewGrid(CCmdUI* pCmdUI);
    afx_msg void OnViewEarth();
    afx_msg void OnUpdateViewEarth(CCmdUI* pCmdUI);
    afx_msg void OnViewPalette();
    afx_msg void OnOptionsEnv();
    afx_msg void OnFileOpen();
    afx_msg void OnFileSave();
    afx_msg void OnFileSaveAs();
    afx_msg void OnEditMode();
    afx_msg void OnUpdateEditMode(CCmdUI* pCmdUI);
    afx_msg void OnViewMapOnly();
    afx_msg void OnUpdateViewMapOnly(CCmdUI* pCmdUI);
    afx_msg void OnViewGraph();

```

```

afx_msg void OnUpdateViewGraph(CCmdUI* pCmdUI);
afx_msg void OnEditTileCoord();
afx_msg void OnUpdateEditTileCoord(CCmdUI* pCmdUI);
afx_msg void OnViewPause();
afx_msg void OnUpdateViewPause(CCmdUI* pCmdUI);
afx_msg void OnMenuStageTitle();
afx_msg void OnEditStand();
afx_msg void OnUpdateEditStand(CCmdUI* pCmdUI);
afx_msg void OnEditCreateMode();
afx_msg void OnUpdateEditCreateMode(CCmdUI* pCmdUI);
afx_msg void OnMenuRecalcDa();
afx_msg void OnMenuSyncEa();
afx_msg void OnViewNoelev();
afx_msg void OnUpdateViewNoelev(CCmdUI* pCmdUI);
afx_msg void OnMenuStageMusic();
afx_msg void OnViewEaGraph();
afx_msg void OnUpdateViewEaGraph(CCmdUI* pCmdUI);
afx_msg void OnMenuElevation();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_MAPEDDOC_H__7ECC1E0C_9C09_11D1_80E2_288A06C10000__INCLUDED_)

```



```

// MapEdView.cpp : implementation of the CMapEdView class
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "MapEd.h"

#include "MapEdDoc.h"
#include "MapEdView.h"
#include "MapListView.h"
#include "MainFrm.h"
#include "../ResMan.h"
#include "../TileMap.h"
#include "../Stage.h"
#include "GetIntDlg.h"
#include "GetTextDlg.h"

#include "../UC2Ani/DIB.h"
#include "../UC2Ani/DIBPal.h"
#include "../UC2Ani/PhSprite.h"

#include "resource.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan    gResMan;
// CMapEdView

IMPLEMENT_DYNCREATE(CMapEdView, COSBView)

BEGIN_MESSAGE_MAP(CMapEdView, COSBView)
    //{{AFX_MSG_MAP(CMapEdView)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
    ON_WM_LBUTTONDBLCLK()
    ON_WM_RBUTTONDOWN()
    ON_COMMAND(ID_SPRITE_DELETE, OnSpriteDelete)
    ON_COMMAND(ID_SPRITE_ELEVATION, OnSpriteElevation)
    ON_COMMAND(ID_SPRITE_FLIP, OnSpriteFlip)
    ON_COMMAND(ID_SPRITE_OPACITY100, OnSpriteOpacity100)
    ON_COMMAND(ID_SPRITE_OPACITY12, OnSpriteOpacity12)
    ON_COMMAND(ID_SPRITE_OPACITY25, OnSpriteOpacity25)
    ON_COMMAND(ID_SPRITE_OPACITY50, OnSpriteOpacity50)
    ON_COMMAND(ID_SPRITE_VERTICAL, OnSpriteVertical)
    ON_COMMAND(ID_SPRITE_FLIPVERT, OnSpriteFlipvert)
    ON_COMMAND(ID_SPRITE_TYPE, OnSpriteType)
    ON_COMMAND(ID_MENU_ANI_REPEAT, OnMenuAniRepeat)
    //}}

```

```

ON_COMMAND(ID_MENU_ANI_FADE, OnMenuAniFade)
ON_COMMAND(ID_MENU_NOANI, OnMenuNoani)
ON_COMMAND(ID_SPRITE_OPACITY75, OnSpriteOpacity75)
ON_COMMAND(ID_ACTOR_ELEVATION, OnActorElevation)
ON_COMMAND(ID_MENU_ANI_RANDOM, OnMenuAniRandom)
ON_WM_SETCURSOR()
ON_COMMAND(ID_MENU_DA_FR, OnMenuDaFr)
ON_COMMAND(ID_MENU_DA_BL, OnMenuDaBl)
ON_COMMAND(ID_MENU_DA_BR, OnMenuDaBr)
ON_COMMAND(ID_MENU_DA_FL, OnMenuDaFl)
ON_COMMAND(ID_MENU_DA_OPEN, OnMenuDaOpen)
ON_COMMAND(ID_MENU_DA_CLOSED, OnMenuDaClosed)
ON_COMMAND(ID_MENU_EXIT_POINT, OnMenuExitPoint)
ON_COMMAND(ID_MENU_ENTRY_POINT, OnMenuEntryPoint)
//}AFX_MSG_MAP
// Standard printing commands
ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////
// CMapEdView construction/destruction

CMapEdView::CMapEdView()
{
    TRACE0("CMapEdView::CMapEdView()\n");
    m_bMouseCaptured = FALSE;
    m_pCS = NULL; // LBUTTONDOWN
    m_pMS = NULL; // RBUTTONDOWN
    m_wCurResType = 0;

    TRACE(" * sizeof(CDIB) = %d bytes\n", sizeof(CDIB));
    TRACE(" * sizeof(CSprite) = %d bytes\n", sizeof(CSprite));
    TRACE(" * sizeof(CPhasedSprite) = %d bytes\n",
sizeof(CPhasedSprite));
}

CMapEdView::~CMapEdView()
{
    TRACE0("CMapEdView::~CMapEdView()\n");
}

BOOL CMapEdView::PreCreateWindow(CREATESTRUCT& cs)
{
    TRACE0("CMapEdView::PreCreateWindow()\n");
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

////////////////////////////////////
// CMapEdView drawing

void CMapEdView::OnDraw(CDC* pDC)
{

```

```

    CMapEdDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    COSBView::OnDraw(pDC);

    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;
    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;
    if (pDoc->IsViewGraph())
        pTM->DrawElevationGraph(pDC);
    if (pDoc->IsViewEAGraph())
        pTM->DrawActorElevationGraph(pDC);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMapEdView printing

BOOL CMapEdView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CMapEdView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CMapEdView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMapEdView diagnostics

#ifdef _DEBUG
void CMapEdView::AssertValid() const
{
    CView::AssertValid();
}

void CMapEdView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CMapEdDoc* CMapEdView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMapEdDoc)));
    return (CMapEdDoc*)m_pDocument;
}
#endif // _DEBUG

```

```

/////////////////////////////////////////////////////////////////
// CMapEdView message handlers

// Create a new buffer and palette to match a new background DIB
BOOL CMapEdView::CreateOSB(CDIB* pDIB)
{
    m_pCS = NULL;
    m_pMS = NULL;

    // Create a new buffer and palette
    if (!COSBView::CreateOSB(pDIB))
        return FALSE;

    // Map the colors of the background DIB to
    // the identity palette we just created for the background
    pDIB->MapColorsToPalette((CPalette*)GetOSBPalette());

    // Resize the main frame window to fit the background image
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit(FALSE); // Try shrinking first
    ResizeParentToFit(TRUE);  // Let's be daring

    Render(); // Render the entire scene to the off-screen buffer
    // DrawOSB(); // Paint the off-screen buffer to the window
    Invalidate(FALSE);

    return TRUE;
}

// Render the scene to the off-screen buffer pClipRect defaults to NULL
void CMapEdView::Render(CRect* pClipRect)
{
    CStage* pStage = GetDocument()->GetStage();
    if (pStage)
        pStage->Render(pClipRect);
}

void CMapEdView::OnLButtonDown(UINT nFlags, CPoint point)
{
    if (m_bMouseCaptured)
        return;

    CClientDC dc(this);
    OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
    CPoint lpt(point);
    dc.DPtoLP(&lpt);

    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;

    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;
}

```

```

// See if it hit a sprite
CPhasedSprite* pSprite;
if (pDoc->IsTileEditMode())
{
    pSprite = (CPhasedSprite*)pStage->TileHitTest(lpt);
}
else
{
    WORD wRM = pStage->GetRenderMode();
    if ((wRM & RM_ELEV) != RM_ELEV) // No Elevation mode
    {
        if (!pDoc->IsStandEditMode()) // OnLButtonDownDblClk
        {
            AfxMessageBox("You can edit only tiles in No
Elevation Mode!");
        }
        return;
    }

    pSprite = (CPhasedSprite*)pStage->SpriteHitTest(lpt);
    if (pSprite && (pSprite->GetSrcType() == SPRITE_TILE))
    {
        pSprite = NULL;
    }
}

if (pSprite)
{
    m_bMouseCaptured = TRUE;
    if (m_pCS != pSprite)
    {
        m_pCS = pSprite;
        TRACE("Sprite %8.8XH Selected(%04x)\n", m_pCS, m_pCS-
>GetType());
    }
    SetCapture();
    m_wCurResType = m_pCS->GetSrcType();

    CSize sEP(m_pCS->GetEarthPoint());
    m_sOffset = lpt - sEP;
    ShowInfo(lpt);
}
//
else if (pDoc->IsCreateMode()) // Insert New Sprite
{
    CMapListView* pMLV = pDoc->GetMapListView();
    ASSERT(pMLV);

    m_wCurResType = pMLV->GetSelectedResType();
    if (m_wCurResType == 0)
        return;
    if (pDoc->IsTileEditMode() &&
        ((m_wCurResType & SPRITE_SRC_MASK) != SPRITE_TILE))
    {
        // Change Edit Mode
        pDoc->SetEditMode(SPRITE_EDIT_MODE);
        return;
    }
}
//

```

```

CString strName;
if (pMLV->GetSelectedResName(strName) < 0)
    return;

// Resource Allocation via Resource Manager
CPhasedSprite* pPS = gResMan.LoadPhasedSprite(strName,
                                                (m_wCurResType !=
SPRITE_ACTOR));
if (!pPS)
{
    strName += " not found!";
    AfxMessageBox(strName);
    return;
}
if ((m_wCurResType == SPRITE_PHASED) ||
    (m_wCurResType == SPRITE_ACTOR))
{
    pPS->SetType(m_wCurResType, SPRITE_ANI_REPEAT);
    pPS->SetMSPT(150L);
}
pPS->SetCell(pMLV->m_nCellID);

if (pDoc->IsTileEditMode())
{
    pPS->MoveToCenter(lpt);
    pStage->InsertTile(pPS);          // AddDirtyRegion
    WORD wRM = pStage->GetRenderMode();
    if ((wRM & RM_ELEV) != RM_ELEV)    // No Elevation mode
    {
        int nElev = pPS->GetElevation();
        if (nElev)
        {
            CRect rc;
            pPS->GetRect(rc);
            rc.top      += nElev;
            rc.bottom   += nElev;
            AddDirtyRegion(&rc);
        }
    }
}
else
{
    CPoint ptTID(pTM->GetNearestTileIndex(lpt));
    CPoint ptC = pTM->GetCenter(ptTID);
    int nElev = pTM->GetElevation(ptTID);
    if ((m_wCurResType & SPRITE_SRC_MASK) != SPRITE_WALL)
    {
        if ((m_wCurResType & SPRITE_ACTOR) == SPRITE_ACTOR)
            pPS->SetElevation(pTM->GetEA(ptTID));
        else
            pPS->SetElevation(nElev);    // Normal Sprites
    }
    pPS->MoveToEarth(pDoc->IsTileCoordMode() ? ptC : lpt);

    if (pPS->GetSrcType() == SPRITE_TILE)    // In sprite edit
mode

```

```

        pPS->SetSrcType(SPRITE_STATIC);        // We put tile as
a normal sprite
        pPS->SetZByEarth();
        pStage->InsertSprite(pPS);        // AddDirtyRegion

// we need to put more walls for elevated tile...
        if (nElev && ((m_wCurResType & SPRITE_SRC_MASK) ==
SPRITE_WALL))
        {
            int n = nElev / ELEVATION_UNIT;
            for (int e=1; e < n; e++)
            {
                CPhasedSprite* pWall =
gResMan.LoadPhasedSprite(strName);
                if (!pWall)
                {
                    strName += " not found!";
                    AfxMessageBox(strName);
                    return;
                }
                pWall->SetCell(pMLV->m_nCellID);
                pWall->SetElevation(e * ELEVATION_UNIT);
                pWall->MoveToEarth(ptC);
                pWall->SetZByEarth();
                pStage->InsertSprite(pWall);    // AddDirtyRegion
            }
        }
        RenderAndDrawDirtyList();
//        ShowInfo(lpt);
    }
//    COSBView::OnLButtonDown(nFlags, point);
}

void CMapEdView::OnLButtonUp(UINT nFlags, CPoint point)
{
    if (m_bMouseCaptured)
    {
        ::ReleaseCapture();
        m_bMouseCaptured = FALSE;
        m_pCS = NULL;
        m_pMS = NULL;
    }
//    COSBView::OnLButtonUp(nFlags, point);
}

void CMapEdView::OnMouseMove(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
    CPoint lpt(point);
    dc.DPtoLP(&lpt);

    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)

```

```

        return;
    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;

    CPoint ptTID(pTM->GetNearestTileIndex(lpt));
    ShowInfo(lpt, ptTID);

    if (!m_bMouseCaptured)
        return;
    ASSERT(m_pCS);

    if (pDoc->IsTileEditMode())
    {
        pStage->MoveTileSpriteTo(m_pCS, lpt);    // Render
    }
    else // Sprite edit mode
    {
        if (pDoc->IsTileCoordMode())
        {
            CPoint ptC(pTM->GetCenter(ptTID));
            if ((m_wCurResType & SPRITE_ACTOR) == SPRITE_ACTOR)
            {
                m_pCS->SetElevation(pTM->GetEA(ptTID));    // for
                switch (pTM->GetDA(ptTID))
                {
                    case DA_OPEN:    m_pCS->SetOpacity(OPACITY_100);
                    break;
                    case DA_CLOSED:  m_pCS->SetOpacity(OPACITY_25);
                    break;
                    default:          m_pCS->SetOpacity(OPACITY_100);
                }
            }
            else if ((m_wCurResType & SPRITE_WALL) != SPRITE_WALL)
            {
                m_pCS->SetElevation(pTM->GetElevation(ptTID));    //
            }
            m_pCS->MoveToEarth(ptC);
        }
        else // Free coord. mode
        {
            CPoint ptNew(lpt - m_sOffset);
            m_pCS->MoveToEarth(ptNew);
        }
        m_pCS->SetZByEarth();
        // For sprites inserted into CSpriteList, SetZ() adds dirty region for
        // itself.
    }
    RenderAndDrawDirtyList();
    // GetDocument()->SetModifiedFlag(TRUE);
    // COSBView::OnMouseMove(nFlags, point);
}

void CMapEdView::ShowInfo(const CPoint& point, const CPoint& ptTID)

```



```

{
    // Show current state in the status bar
    CMainFrame* pFrame = (CMainFrame*)(AfxGetApp()->m_pMainWnd);
    ASSERT(pFrame);
    int nS=0;
    int nA=0;
    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (pStage)
    {
        nS = pStage->GetNumSprites();
        nA = pStage->GetNumAniSprites();
    }

    char buf[256];
    wsprintf(buf, "(%d,%d) T[%d,%d] Sprites:%d Ani:%d",
        point.x, point.y, ptTID.x, ptTID.y, nS, nA);
    pFrame->SetStatusBarText(buf);
}

void CMapEdView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
    CPoint lpt(point);
    dc.DPtoLP(&lpt);

    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;
    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;

    if (pDoc->IsStandEditMode())
    {
        m_ptHit = lpt;
        OnMenuDirectionAttribute(DA_CLOSED);
        return;
    }

    // See if it hit a sprite
    CPhasedSprite* pSprite;
    if (pDoc->IsTileEditMode())
    {
        pSprite = (CPhasedSprite*)pStage->TileHitTest(lpt);
        if (pSprite)
        {
            CPoint ptTID(pTM->GetNearestTileIndex(lpt));
            pTM->InCEA(ptTID, ELEVATION_UNIT);
        }
    }
    else
    {
        pSprite = (CPhasedSprite*)pStage->SpriteHitTest(lpt);
    }
}

```

```

        if (pSprite && (pSprite->GetSrcType() == SPRITE_TILE))
        {
            pSprite = NULL;
        }
    }

    if (pSprite)
    {
        pSprite->IncElevation(ELEVATION_UNIT);
        pSprite->MoveBy(0, -ELEVATION_UNIT);
        pSprite->SetZByEarth();
        if (pDoc->IsViewGraph())
            Invalidate(FALSE);        // To update DrawElevationGraph()
        RenderAndDrawDirtyList();
    }
    COSBView::OnLButtonDblClk(nFlags, point);
}

void CMapEdView::OnRButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc); // CScrollView changes the viewport origin and
mapping mode.
    CPoint lpt(point);
    dc.DPtoLP(&lpt);

    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;

    if (pDoc->IsStandEditMode())
    {
        CTileMap* pTM = pStage->GetTileMap();
        if (!pTM)
            return;

        CPoint ptTID(pTM->GetNearestTileIndex(lpt));

        CMenu menu;
        VERIFY(menu.LoadMenu(IDR_MENU_GRID));
        CMenu* pPopup = menu.GetSubMenu(0);
        m_ptHit = lpt;

        WORD wDA = pTM->GetDA(ptTID);
        if ((wDA & DA_FR) == DA_FR)
            pPopup->CheckMenuItem(ID_MENU_DA_FR, MF_CHECKED);
        if ((wDA & DA_FL) == DA_FL)
            pPopup->CheckMenuItem(ID_MENU_DA_FL, MF_CHECKED);
        if ((wDA & DA_BR) == DA_BR)
            pPopup->CheckMenuItem(ID_MENU_DA_BR, MF_CHECKED);
        if ((wDA & DA_BL) == DA_BL)
            pPopup->CheckMenuItem(ID_MENU_DA_BL, MF_CHECKED);
        if (wDA == DA_OPEN)
            pPopup->CheckMenuItem(ID_MENU_DA_OPEN, MF_CHECKED);
        else if (wDA == DA_CLOSED)
            pPopup->CheckMenuItem(ID_MENU_DA_CLOSED, MF_CHECKED);
    }
}

```

```

        ClientToScreen(&point);
        pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
point.y, this);
        return;
    }

    // See if it hit a sprite
    CPhasedSprite* pSprite;
    if (pDoc->IsTileEditMode())
    {
        pSprite = (CPhasedSprite*)pStage->TileHitTest(lpt);
    }
    else
    {
        pSprite = (CPhasedSprite*)pStage->SpriteHitTest(lpt);
        if (pSprite && (pSprite->GetSrcType() == SPRITE_TILE))
        {
            pSprite = NULL;
        }
    }
    if (!pSprite)
        return;

    CMenu menu;
    VERIFY(menu.LoadMenu(IDR_MENU_TILE));
    CMenu* pPopup = menu.GetSubMenu(0);
    ASSERT(pPopup);
    m_pMS = pSprite;
    m_ptHit = lpt;

    CString strName(*pSprite->GetName());
    gResMan.MakeResName(strName);
    CString strZ;
    strZ.Format("[%d]", pSprite->GetZ());
    strName += strZ;
    CString strType;
    switch (pSprite->GetSrcType())
    {
        case SPRITE_TILE:         strType = "Tile"; break;
        case SPRITE_WALL:         strType = "Wall"; break;
        case SPRITE_STATIC:       strType = "Sprite"; break;
        case SPRITE_PHASED:       strType = "Phased"; break;
        case SPRITE_ACTOR:        strType = "Actor"; break;
    }
    switch (pSprite->GetAniType())
    {
        case SPRITE_ANI_REPEAT: strType += "(Repeat)"; break;
        case SPRITE_ANI_FADE:   strType += "(Fade)"; break;
        case SPRITE_ANI_ACTOR:  strType += "(Actor)"; break;
    }
    strName += strType;
    pPopup->InsertMenu(0, MF_BYPOSITION | MF_STRING, ID_SPRITE_TYPE,
strName);
    CString strText;
    strText.Format("Elev=%d", pSprite->GetElevation());
    pPopup->ModifyMenu(ID_SPRITE_ELEVATION, MF_BYCOMMAND | MF_STRING,

```

```

                                ID_SPRITE_ELEVATION, strText);
if (pSprite->GetSrcType() == SPRITE_TILE)
{
    CTileMap* pTM = pStage->GetTileMap();
    if (pTM)
    {
        CPoint ptTID(pTM->GetNearestTileIndex(lpt));
        strText.Format("A.Elev=%d", pTM->GetEA(ptTID));
        pPopup->InsertMenu(ID_SPRITE_ELEVATION, MF_BYCOMMAND |
MF_STRING,
                                ID_ACTOR_ELEVATION, strText);
        if (pSprite->HasHyperlink())
        {
            if (pSprite->GetLinkType() ==
CPhasedSprite::HLINK_U2_ENTRY)
            {
                pPopup->ModifyMenu(ID_MENU_ENTRY_POINT,
MF_BYCOMMAND | MF_STRING,
                                ID_MENU_ENTRY_POINT, *pSprite->GetHyperlink());
                pPopup->CheckMenuItem(ID_MENU_ENTRY_POINT,
MF_CHECKED);
            }
            else
            {
                pPopup->ModifyMenu(ID_MENU_EXIT_POINT,
MF_BYCOMMAND | MF_STRING,
                                ID_MENU_EXIT_POINT, *pSprite->GetHyperlink());
                pPopup->CheckMenuItem(ID_MENU_EXIT_POINT,
MF_CHECKED);
            }
        }
    }
    else
    {
        pPopup->RemoveMenu(ID_MENU_ENTRY_POINT, MF_BYCOMMAND |
MF_STRING);
        pPopup->RemoveMenu(ID_MENU_EXIT_POINT, MF_BYCOMMAND | MF_STRING);
    }

    if (pSprite->IsImageFlip())
        pPopup->CheckMenuItem(ID_SPRITE_FLIP, MF_CHECKED);
    if (pSprite->IsImageVertical())
        pPopup->CheckMenuItem(ID_SPRITE_VERTICAL, MF_CHECKED);
    if (pSprite->IsImageFlip() && pSprite->IsImageVertical())
        pPopup->CheckMenuItem(ID_SPRITE_FLIPVERT, MF_CHECKED);

    switch (pSprite->GetAniType())
    {
    case SPRITE_ANI_REPEAT:
        pPopup->CheckMenuItem(ID_MENU_ANI_REPEAT, MF_CHECKED);
        break;
    case SPRITE_ANI_FADE:
        pPopup->CheckMenuItem(ID_MENU_ANI_FADE, MF_CHECKED); break;
    case SPRITE_ANI_RANDOM:

```

```

        pPopup->CheckMenuItem(ID_MENU_ANI_RANDOM, MF_CHECKED);
    break;
}

WORD wImOp = pSprite->GetImOp();
switch (wImOp & OPACITY_MASK)
{
    case OPACITY_100:
        pPopup->CheckMenuItem(ID_SPRITE_OPACITY100, MF_CHECKED);
    break;
    case OPACITY_75:
        pPopup->CheckMenuItem(ID_SPRITE_OPACITY75, MF_CHECKED);
    break;
    case OPACITY_50:
        pPopup->CheckMenuItem(ID_SPRITE_OPACITY50, MF_CHECKED);
    break;
    case OPACITY_25:
        pPopup->CheckMenuItem(ID_SPRITE_OPACITY25, MF_CHECKED);
    break;
    case OPACITY_12:
        pPopup->CheckMenuItem(ID_SPRITE_OPACITY12, MF_CHECKED);
    break;
}

ClientToScreen(&point);
pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
point.y, this);

// COSBView::OnRButtonDown(nFlags, point);
}

void CMapEdView::OnSpriteDelete()
{
    if (!m_pMS)
        return;
    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;
    if (pDoc->IsTileEditMode())
        pStage->DeleteTile(m_pMS);
    else
        pStage->DeleteSprite(m_pMS);
    m_pCS = NULL;
    m_pMS = NULL;
    RenderAndDrawDirtyList();
}

////////////////////////////////////
// Elevation
void CMapEdView::OnSpriteElevation()
{
    if (!m_pMS)
        return;
    CMapEdDoc* pDoc = GetDocument();

    int nElevOld = m_pMS->GetElevation();

```

```

CGetIntDlg dlg;
dlg.m_strCaption.LoadString(IDS_ENTER_ELEVATION);
dlg.m_nNew = nElevOld; // / ELEVATION_UNIT;
if (dlg.DoModal() != IDOK)
    return;

m_pMS->SetElevation(dlg.m_nNew); // * ELEVATION_UNIT;
m_pMS->MoveBy(0, nElevOld - dlg.m_nNew); // * ELEVATION_UNIT;
m_pMS->SetZByEarth();

if (m_pMS->GetSrcType() == SPRITE_TILE)
{
    CStage* pStage = pDoc->GetStage();
    if (pStage)
    {
        CTileMap* pTM = pStage->GetTileMap();
        if (pTM)
        {
            // call GetNearestTileIndex() after MoveBy
            CPoint ptTID(pTM->GetNearestTileIndex(m_pMS));
            pTM->SetEA(ptTID, m_pMS->GetElevation());
        }
    }
}
RenderAndDrawDirtyList();
if (pDoc->IsViewGraph())
    Invalidate(FALSE); // To update DrawElevationGraph()
}

void CMapEdView::OnActorElevation()
{
    if (!m_pMS)
        return;
    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;
    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;

    // m_pMS is tile
    CPoint ptTID(pTM->GetNearestTileIndex(m_pMS));

    CGetIntDlg dlg;
    dlg.m_strCaption.LoadString(IDS_ENTER_ACTOR_ELEVATION);
    dlg.m_nNew = pTM->GetEA(ptTID);
    if (dlg.DoModal() != IDOK)
        return;

    pTM->SetEA(ptTID, dlg.m_nNew);
    if (pDoc->IsViewEAGraph())
        Invalidate(FALSE); // To update DrawElevationGraph()
}

void CMapEdView::ChangeSpriteImOp(const int nMenu)
{

```

```

    if (!m_pMS)
        return;

    WORD wImOp = m_pMS->GetImOp();
    switch (nMenu)
    {
    case ID_SPRITE_FLIP:
        wImOp ^= IMAGE_FLIP;          break;
    case ID_SPRITE_VERTICAL:
        wImOp ^= IMAGE_VERTICAL;      break;
    case ID_SPRITE_FLIPVERT:
        if (m_pMS->IsImageFlip() && m_pMS->IsImageVertical())
            wImOp &= ~ORIENT_MASK;
        else
            wImOp |= (IMAGE_FLIP | IMAGE_VERTICAL); break;
    case ID_SPRITE_OPACITY100:
        wImOp = (wImOp & ~OPACITY_MASK) | OPACITY_100;
    break;
    case ID_SPRITE_OPACITY75:
        wImOp = (wImOp & ~OPACITY_MASK) | OPACITY_75;
    break;
    case ID_SPRITE_OPACITY50:
        wImOp = (wImOp & ~OPACITY_MASK) | OPACITY_50;
    break;
    case ID_SPRITE_OPACITY25:
        wImOp = (wImOp & ~OPACITY_MASK) | OPACITY_25;
    break;
    case ID_SPRITE_OPACITY12:
        wImOp = (wImOp & ~OPACITY_MASK) | OPACITY_12;
    break;
    }
    m_pMS->SetImOp(wImOp);

    AddDirtyRegion(m_pMS);
    RenderAndDrawDirtyList();
}

void CMapEdView::OnSpriteFlip()
{
    ChangeSpriteImOp(ID_SPRITE_FLIP);
}
void CMapEdView::OnSpriteVertical()
{
    ChangeSpriteImOp(ID_SPRITE_VERTICAL);
}
void CMapEdView::OnSpriteFlipvert()
{
    ChangeSpriteImOp(ID_SPRITE_FLIPVERT);
}

void CMapEdView::OnSpriteOpacity100()
{
    ChangeSpriteImOp(ID_SPRITE_OPACITY100);
}
void CMapEdView::OnSpriteOpacity75()
{
    ChangeSpriteImOp(ID_SPRITE_OPACITY75);
}
void CMapEdView::OnSpriteOpacity50()
{
    ChangeSpriteImOp(ID_SPRITE_OPACITY50);
}
void CMapEdView::OnSpriteOpacity25()
{
    ChangeSpriteImOp(ID_SPRITE_OPACITY25);
}
void CMapEdView::OnSpriteOpacity12()
{
    ChangeSpriteImOp(ID_SPRITE_OPACITY12);
}

/*
void CMapEdView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

```

```

{
    // switch for input key
    switch (nChar)
    {
        case VK_UP:      pDoc->SendCommand(CMD_MOVE_NORTH); break;
        case VK_LEFT:    pDoc->SendCommand(CMD_MOVE_WEST); break;
        case VK_DOWN:    pDoc->SendCommand(CMD_MOVE_SOUTH); break;
        case VK_RIGHT:   pDoc->SendCommand(CMD_MOVE_EAST); break;
        // case VK_F5:    SetMode(0); break;
        // case VK_F6:    SetMode(1); break;
        // case VK_F7:    SetMode(1); break;
        // case VK_F8:    SetMode(2); break;
        // case VK_F9:    SetMode(3); break;
        // case VK_F10:   SetMode(4); break;
    }

    // Show current state in the status bar
    // CMainFrame* pFrame = (CMainFrame*)(AfxGetApp()->m_pMainWnd);
    // ASSERT(pFrame);
    // pFrame->m_wndStatusBar.SetWindowText(buf);

    COSBView::OnKeyDown(nChar, nRepCnt, nFlags);
}
*/

void CMapEdView::OnSpriteType()
{
    CMapEdDoc* pDoc = GetDocument();
    CMapListView* pMLV = pDoc->GetMapListView();
    ASSERT(pMLV);

    if (!m_pMS)
        return;
    CString strName(*m_pMS->GetName());
    gResMan.MakeResName(strName);
    int i = gResMan.GetSpriteID(strName);
    if (i >= 0)
        pMLV->SetSelectedItem(i);
}

void CMapEdView::AnimateSprite(CPhasedSprite* pPS, const WORD wAniType)
{
    CStage* pStage = GetDocument()->GetStage();
    if (!pStage)
        return;

    CGetIntDlg dlg;
    dlg.m_strCaption.LoadString(IDS_ENTER_MSPT);
    dlg.m_nNew = pPS->GetMSPT() ? pPS->GetMSPT() : 100;

    if (dlg.DoModal() != IDOK)
        return;
    pPS->SetMSPT(dlg.m_nNew); // includes SetAlarmTick();
    pPS->SetAniType(wAniType);
    pStage->InsertAniSprite(pPS);
}

void CMapEdView::OnMenuAniRepeat()

```



```

{
    if (!m_pMS)
        return;
    AnimateSprite(m_pMS, SPRITE_ANI_REPEAT);
}

void CMapEdView::OnMenuAniFade()
{
    if (!m_pMS)
        return;
    AnimateSprite(m_pMS, SPRITE_ANI_FADE);
}

void CMapEdView::OnMenuAniRandom()
{
    if (!m_pMS)
        return;
    AnimateSprite(m_pMS, SPRITE_ANI_RANDOM);
}

void CMapEdView::OnMenuNoani()
{
    if (!m_pMS)
        return;

    CStage* pStage = GetDocument()->GetStage();
    if (!pStage)
        return;
    // CString strName(*m_pMS->GetName());
    // m_pMS->SetType(gResMan.GetSpriteTypeByName(strName)); // Restore
    m_pMS->SetAniType(0);
    pStage->RemoveAniSprite(m_pMS); // Multiple calls possible
}

void CMapEdView::AdjustFrameWindow()
{
    CMainFrame* pMF = (CMainFrame*)GetParentFrame();
    ASSERT(pMF);
    // pMF->SetFreeze(FALSE);
    pMF->SetPalette((CPalette*)GetOSBPalette());

    return;

    /*
    CDIB* pDIB = GetOSB();
    if (pDIB)
    {
        CClientDC dc(this);
        int nWidth = min(dc.GetDeviceCaps(HORZRES) - 40, pDIB-
>GetWidth());
        int nHeight = min(dc.GetDeviceCaps(VERTRES) - 40, pDIB-
>GetHeight());
        if (((pDIB->GetWidth() + 40) >= dc.GetDeviceCaps(HORZRES)) ||
            ((pDIB->GetHeight() + 40) >= dc.GetDeviceCaps(VERTRES)))
            pMF->SetWindowPos(&wndTopMost, 0, 0, nWidth, nHeight,
SWP_NOZORDER);
        else
    */

```

```

        pMF->SetWindowPos(&wndTopMost, 0, 0, nWidth, nHeight,
SWP_NOMOVE | SWP_NOZORDER);
    }
    // Resize the main frame window to fit the background image
    pMF->RecalcLayout();
    ResizeParentToFit(FALSE);    // Try shrinking first
    ResizeParentToFit(TRUE);     // Let's be daring
    // pMF->SetFreeze();
    /*
}

void CMapEdView::OnMenuEntryPoint()
{
    SetEntryExitPoint(TRUE);
}

void CMapEdView::OnMenuExitPoint()
{
    SetEntryExitPoint(FALSE);
}

void CMapEdView::SetEntryExitPoint(const BOOL bEntry)
{
    if (!m_pMS)
        return;
    extern const char* STAGEFILE_FILTER;

    CString strFilename(*m_pMS->GetHyperlink());
    ::SetCurrentDirectory(*gResMan.GetResPath());
    // Show a File Save dialog to get the name.
    CFileDialog dlg(TRUE,    // Open
                    NULL,    // No default extension
                    strFilename,    // initial file name
                    OFN_OVERWRITEPROMPT | OFN_HIDEREADONLY,
                    STAGEFILE_FILTER);
    if (dlg.DoModal() == IDOK)
    {
        strFilename = dlg.GetPathName();
        if (!strFilename.IsEmpty())
        {
            gResMan.MakeResName(strFilename);
            CString strHL;
            strHL.Format("%c:%s", bEntry ? 'e' : 'x', strFilename);
            m_pMS->SetHyperlink(strHL);
        }
    }
    else
    {
        if (!strFilename.IsEmpty())
            m_pMS->GetHyperlink()->Empty();    // Erase
    }
    if (GetDocument()->IsViewGraph())
        Invalidate(FALSE);    // To update DrawElevationGraph()
}
/*
CGetTextDlg dlg;
dlg.m_strCaption.LoadString(IDS_ENTER_STAGE_LINK);
CString* pStr = m_pMS->GetHyperlink();

```

```

        if (!pStr->IsEmpty())
            dlg.m_strText = *pStr;
        if (dlg.DoModal() == IDOK)
        {
            if (!dlg.m_strText.IsEmpty())
                m_pMS->SetHyperlink(dlg.m_strText);
        }
    */
}

BOOL CMapEdView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    CMapEdDoc* pDoc = GetDocument();
    if (pDoc->IsTileEditMode())
    {
        ::SetCursor(AfxGetApp()->LoadCursor(m_bMouseCaptured ?
        IDC_GREP_DOWN : IDC_GREP_UP));
        return TRUE;
    }
    else
        return COSBView::OnSetCursor(pWnd, nHitTest, message);
}

// On Direction Attribute: Forward Right
void CMapEdView::OnMenuDaFr() { OnMenuDirectionAttribute(DA_FR); }
void CMapEdView::OnMenuDaBl() { OnMenuDirectionAttribute(DA_BL); }
void CMapEdView::OnMenuDaBr() { OnMenuDirectionAttribute(DA_BR); }
void CMapEdView::OnMenuDaFl() { OnMenuDirectionAttribute(DA_FL); }
void CMapEdView::OnMenuDaOpen() {
    OnMenuDirectionAttribute(DA_OPEN); }
void CMapEdView::OnMenuDaClosed() { OnMenuDirectionAttribute(DA_CLOSED); }

void CMapEdView::OnMenuDirectionAttribute(const WORD wDA)
{
    CMapEdDoc* pDoc = GetDocument();
    CStage* pStage = pDoc->GetStage();
    if (!pStage)
        return;
    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;

    CPoint ptTID(pTM->GetNearestTileIndex(m_ptHit));
    CPoint ptCtr(pTM->GetNearestTileCenter(m_ptHit));

    WORD wDirAttr;
    if ((wDA == DA_OPEN) || (wDA == DA_CLOSED)) // Assign
    {
        wDirAttr = wDA;
    }
    else
    {
        wDirAttr = pTM->GetDA(ptTID);
        wDirAttr ^= wDA; // toggle only the specified bit
    }

    pTM->SetDA(ptTID, wDirAttr);
}

```

```

    CRect rcDirty;
    rcDirty.SetRect(ptCtr.x-64, ptCtr.y-32, ptCtr.x+64, ptCtr.y+32);
    AddDirtyRegion(&rcDirty);
    RenderAndDrawDirtyList();
}

void CMapEdView::RecalculatedA()
{
    CStage* pStage = GetDocument()->GetStage();
    if (!pStage)
        return;
    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;
    pTM->RecalculatedA();
}

void CMapEdView::SynchronizeEA()
{
    CStage* pStage = GetDocument()->GetStage();
    if (!pStage)
        return;
    CTileMap* pTM = pStage->GetTileMap();
    if (!pTM)
        return;
    pTM->SynchronizeAllEA();
}

```

MapEd\MapEdView.h

```
// MapEdView.h : interface of the CMapEdView class
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#if
!defined(AFX_MAPEDVIEW_H__7ECC1E0E_9C09_11D1_80E2_288A06C10000__INCLUDED_)
#define AFX_MAPEDVIEW_H__7ECC1E0E_9C09_11D1_80E2_288A06C10000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "../UC2Ani/OSBView.h"

class CSprite;
class CPhasedSprite;
class CMapEdDoc;
class CStage;

class CMapEdView : public COSBView
{
protected: // create from serialization only
    CMapEdView();
    DECLARE_DYNCREATE(CMapEdView)

// Attributes
public:
    CMapEdDoc* GetDocument();

// Operations
public:
    virtual BOOL        CreateOSB(CDIB* pDIB);           // COSBView
    virtual void        Render(CRect* pClipRect=NULL);   // COSBView
    CPhasedSprite*      GetCapturedSprite() { return m_pCS; }
    void                AdjustFrameWindow();
    void                RecalculateDA();
    void                SynchronizeEA();

private:
    void SetEntryExitPoint(const BOOL bEntry);
    void ShowInfo(const CPoint& point, const CPoint& ptTID);

    BOOL        m_bMouseCaptured; // TRUE if mouse captured
    CPhasedSprite* m_pCS;           // pointer to captured
sprite (for drag)
    CSize        m_sOffset;         // offset into hit sprite
    CPhasedSprite* m_pMS;           // pointer to current
sprite
    CPoint        m_ptHit;          // Hit point
    WORD          m_wCurResType;   // Current Resource Type

// Overrides
    // ClassWizard generated virtual function overrides

```

```

//{{AFX_VIRTUAL(CMapEdView)
public:
    virtual void OnDraw(CDC* pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMapEdView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    void ChangeSpriteImOp(const int nMenu);
    void AnimateSprite(CPhasedSprite* pS, const WORD wAniType);
    void OnMenuDirectionAttribute(const WORD wDA);

//{{AFX_MSG(CMapEdView)
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
afx_msg void OnMouseMove(UINT nFlags, CPoint point);
afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
afx_msg void OnSpriteDelete();
afx_msg void OnSpriteElevation();
afx_msg void OnSpriteFlip();
afx_msg void OnSpriteOpacity100();
afx_msg void OnSpriteOpacity12();
afx_msg void OnSpriteOpacity25();
afx_msg void OnSpriteOpacity50();
afx_msg void OnSpriteVertical();
afx_msg void OnSpriteFlipvert();
afx_msg void OnSpriteType();
afx_msg void OnMenuAniRepeat();
afx_msg void OnMenuAniFade();
afx_msg void OnMenuNoani();
afx_msg void OnSpriteOpacity75();
afx_msg void OnActorElevation();
afx_msg void OnMenuAniRandom();
afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
afx_msg void OnMenuDaFr();
afx_msg void OnMenuDaBl();
afx_msg void OnMenuDaBr();
afx_msg void OnMenuDaFl();
afx_msg void OnMenuDaOpen();
afx_msg void OnMenuDaClosed();
afx_msg void OnMenuExitPoint();
afx_msg void OnMenuEntryPoint();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

```

```
};
```

```
#ifndef _DEBUG // debug version in MapEdView.cpp
inline CMapEdDoc* CMapEdView::GetDocument()
{ return (CMapEdDoc*)m_pDocument; }
#endif
```

```
////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
```

```
#endif //
#define(AFX_MAPEDVIEW_H__7ECC1E0E_9C09_11D1_80E2_288A06C10000__INCLUDED_)
```

```

// MapEnvDlg.cpp : implementation file
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "MapEd.h"
#include "MapEnvDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CMapEnvDlg dialog

CMapEnvDlg::CMapEnvDlg(CWnd* pParent /*=NULL*/)
: CDialog(CMapEnvDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CMapEnvDlg)
    m_nTileHeight = 0;
    m_nTileWidth = 0;
    m_nScrHeight = 0;
    m_nScrWidth = 0;
    m_strDataPath = _T("");
    m_strPalFile = _T("");
    //}}AFX_DATA_INIT
    m_bReadOnly = FALSE;
}

void CMapEnvDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMapEnvDlg)
    DDX_Text(pDX, IDC_EDIT_TILE_H, m_nTileHeight);
    DDV_MinMaxInt(pDX, m_nTileHeight, 4, 480);
    DDX_Text(pDX, IDC_EDIT_TILE_W, m_nTileWidth);
    DDV_MinMaxInt(pDX, m_nTileWidth, 4, 640);
    DDX_Text(pDX, IDC_EDIT_SCREEN_H, m_nScrHeight);
    DDV_MinMaxInt(pDX, m_nScrHeight, 32, 768);
    DDX_Text(pDX, IDC_EDIT_SCREEN_W, m_nScrWidth);
    DDV_MinMaxInt(pDX, m_nScrWidth, 64, 1024);
    DDX_Text(pDX, IDC_EDIT_DATA_PATH, m_strDataPath);
    DDX_Text(pDX, IDC_EDIT_PALFILE, m_strPalFile);
    DDV_MaxChars(pDX, m_strPalFile, 12);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMapEnvDlg, CDialog)
    //{{AFX_MSG_MAP(CMapEnvDlg)

```



```

    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////
// CMapEnvDlg message handlers

```

```

BOOL CMapEnvDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    if (m_bReadOnly)
    {
        GetDlgItem(IDC_EDIT_TILE_W)->EnableWindow(FALSE);
        GetDlgItem(IDC_EDIT_TILE_H)->EnableWindow(FALSE);
        GetDlgItem(IDC_EDIT_SCREEN_W)->EnableWindow(FALSE);
        GetDlgItem(IDC_EDIT_SCREEN_H)->EnableWindow(FALSE);
    }
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

#if
!defined(AFX_MAPENVDLG_H__18238822_A15C_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_MAPENVDLG_H__18238822_A15C_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MapEnvDlg.h : header file
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

////////////////////////////////////
// CMapEnvDlg dialog

class CMapEnvDlg : public CDialog
{
// Construction
public:
    CMapEnvDlg(CWnd* pParent = NULL);    // standard constructor

    void SetReadOnly()    { m_bReadOnly = TRUE; }

// Dialog Data
   //{{AFX_DATA(CMapEnvDlg)
    enum { IDD = IDD_ENVIRONMENT };
    int         m_nTileHeight;
    int         m_nTileWidth;
    int         m_nScrHeight;
    int         m_nScrWidth;
    CString     m_strDataPath;
    CString     m_strPalFile;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMapEnvDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    BOOL m_bReadOnly;

    // Generated message map functions
   //{{AFX_MSG(CMapEnvDlg)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}

```

// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //

!defined(AFX_MAPENVDLG_H__18238822_A15C_11D1_80E2_080009B9F339__INCLUDED_)

```

// MapListView.cpp : implementation file
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "MapEd.h"
#include "MapListView.h"
#include "MapEdView.h"
#include "MapEdDoc.h"
#include "../ResMan.h"
#include "../UC2Ani/PhSprite.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CResMan    gResMan;

////////////////////////////////////
// CMapListView

IMPLEMENT_DYNCREATE(CMapListView, CFormView)

CMapListView::CMapListView()
    : CFormView(CMapListView::IDD)
{
    TRACE0("CMapListView::CMapListView()\n");
    //{{AFX_DATA_INIT(CMapListView)
    m_strCellInfo = _T("");
    m_nCellID = 0;
    //}}AFX_DATA_INIT
    m_ilObject.Create(IDB_IL_OBJECT, 16, 1, CLR_NONE);
    m_nIndex      = -1;
    m_nCellID     = 0;
    m_pPS         = NULL;
    m_bFirst      = TRUE;
}

CMapListView::~CMapListView()
{
    TRACE0("CMapListView::~CMapListView()\n");
    if (m_pPS)
        delete m_pPS;
}

void CMapListView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMapListView)
    DDX_Control(pDX, IDC_LIST_OBJ, m_ilObject);
    DDX_Text(pDX, IDC_ST_CELL_INFO, m_strCellInfo);

```

```

        DDX_Scroll(pDX, IDC_SB_CELL_ID, m_nCellID);
        //}}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CMapListView, CFormView)
    //{{AFX_MSG_MAP(CMapListView)
    ON_NOTIFY(NM_CLICK, IDC_LIST_OBJ, OnClickListObj)
    ON_WM_HSCROLL()
    ON_NOTIFY(NM_DBLCLK, IDC_LIST_OBJ, OnDblclkListObj)
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_KEYDOWN()
    ON_WM_QUERYNEWPALETTE()
    ON_WM_PALETTECHANGED()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CMapListView diagnostics

#ifdef _DEBUG
void CMapListView::AssertValid() const
{
    CFormView::AssertValid();
}

void CMapListView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CMapEdDoc* CMapListView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMapEdDoc)));
    return (CMapEdDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////
// CMapListView message handlers

// CDocument::UpdateAllViews() will call this function
// Multiple calls possible
void CMapListView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();

    if (m_bFirst)
    {
        Initialize();
        m_bFirst = FALSE;
    }
    UpdateList();
}

void CMapListView::Initialize()
{

```

```

// Prepare List Control for Channels list
m_lcObject.SetImageList(&m_ilObject, LVSIL_SMALL);

char* szColumn[] = {"SPRITE", "CxR"};
int    nWidth[]   = {120, 50};

LV_COLUMN  lvC; // list view column structure
lvC.mask    = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM; //
valid members
lvC.fmt      = LVCFMT_LEFT; // left-align column
// Add the columns.
for (int i=0; i < sizeof(nWidth)/sizeof(nWidth[0]); i++)
{
    lvC.cx          = nWidth[i]; // width of column in
pixels
    lvC.iSubItem     = i;
    lvC.pszText      = szColumn[i];
    if (m_lcObject.InsertColumn(i, &lvC) == -1)
        return;
}

CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_CELL_ID);
CRect rc;
pSB->GetWindowRect(&rc);
m_rcDraw = CRect(5, 208, 5+rc.Width(), 480);
pSB->ShowWindow(FALSE);
}

void CMapListView::UpdateList()
{
    // Initialize
    m_lcObject.DeleteAllItems();
    m_nIndex = -1;
    m_nCellID = 0;
    if (m_pPS)
    {
        delete m_pPS;
        m_pPS = NULL;
    }
    InvalidateRect(&m_rcDraw);

    int nSprites = gResMan.GetNumSprites();
    for (int i=0; i < nSprites; i++)
    {
        LV_ITEM  lvI; // list view item structure
        ::ZeroMemory(&lvI, sizeof(lvI));
        lvI.mask    = LVIF_TEXT | LVIF_IMAGE | LVIF_PARAM |
LVIF_STATE;
        lvI.state    = 0;
        lvI.stateMask = 0;

        lvI.iItem     = i;
        lvI.iSubItem   = 0;
        CString* pStr = gResMan.GetSpriteName(i);
        lvI.pszText    = pStr->GetBuffer(17);
        lvI.cchTextMax = 17;
        switch (gResMan.GetSpriteType(i))

```

```

        {
            case SPRITE_TILE: lvI.iImage = 0; break;
            case SPRITE_WALL: lvI.iImage = 1; break;
            case SPRITE_STATIC: lvI.iImage = 2; break;
            case SPRITE_PHASED: lvI.iImage = 3; break;
            case SPRITE_ACTOR: lvI.iImage = 4; break;
        }
        CSize szCell(gResMan.GetSpriteColRow(i));
        lvI.lParam = (LPARAM)(szCell.cx * szCell.cy); // # of
cells
        m_lcObject.InsertItem(&lvI);

        CString strCell;
        strCell.Format("%dx%d", szCell.cx, szCell.cy);
        m_lcObject.SetItemText(i, 1, strCell);
    }
}

// Update Items and Images
void CMapListView::UpdateItemImage(const int nIndex)
{
    int nCells = (int)m_lcObject.GetItemData(nIndex);
    CString strName = m_lcObject.GetItemText(nIndex, 0); // SubItem=0
    m_strCellInfo.Format("%s (%d/%d)", strName, 0, nCells);
    UpdateData(FALSE);
    CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_CELL_ID);
    pSB->SetScrollRange(0, nCells-1);
    pSB->SetScrollPos(0);
    pSB->ShowWindow(nCells > 1); // // Hide for STATIC SPRITE

    // We'll not reuse DIB here:
    // Since this image is not need to be resident in memory.
    CPhasedSprite* pPS = gResMan.LoadPhasedSprite(strName, FALSE);
    if (!pPS)
    {
        delete pPS;
        strName += " not found!";
        AfxMessageBox(strName);
        return;
    }

    if (m_pPS)
        delete m_pPS; // Delete previously allocated resource
    m_pPS = pPS;
    m_nCellID = 0;
    m_pPS->SetCell(m_nCellID);
    if (m_pPS->GetSrcType() == SPRITE_TILE)
        GetDocument()->SetEditMode(TILE_EDIT_MODE);
    else
        GetDocument()->SetEditMode(SPRITE_EDIT_MODE);

    InvalidateRect(&m_rcDraw);
}

void CMapListView::OnClickListObj(NMHDR* pNMHDR, LRESULT* pResult)
{
    DWORD dwPos = ::GetMessagePos();

```

```

    CPoint point((int)LOWORD(dwPos), (int)HIWORD(dwPos));
    m_lcObject.ScreenToClient(&point);

    int nHitIndex = m_lcObject.HitTest(point);
    if (nHitIndex == -1)
        return;
    if (m_nIndex != nHitIndex)
    {
        m_nIndex = nHitIndex;
        UpdateItemImage(m_nIndex);
    }
    *pResult = 0;
}

void CMapListView::OnDblclkListObj(NMHDR* pNMHDR, LRESULT* pResult)
{
    DWORD dwPos = ::GetMessagePos();
    CPoint point((int)LOWORD(dwPos), (int)HIWORD(dwPos));
    m_lcObject.ScreenToClient(&point);

    int nHitIndex = m_lcObject.HitTest(point);
    if (nHitIndex == -1)
        return;
    if (m_nIndex != nHitIndex)
    {
        m_nIndex = nHitIndex;
        UpdateItemImage(m_nIndex);
    }
    *pResult = 0;
}

void CMapListView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_CELL_ID);
    if (pScrollBar == pSB)
    {
        if (m_nIndex < 0)
            return;
        int nCells = (int)m_lcObject.GetItemData(m_nIndex);
        CString strName = m_lcObject.GetItemText(m_nIndex, 0); //
        SubItem=0
        int nMin, nMax;
        pSB->GetScrollRange(&nMin, &nMax);
        switch (nSBCode)
        {
            case SB_THUMBPOSITION:
                m_nCellID = nPos;
                pSB->SetScrollPos(nPos);
                break;
            case SB_LINELEFT:
                m_nCellID = pSB->GetScrollPos() - 1;
                if (m_nCellID < nMin)
                    m_nCellID = nMax; // Wrap
                pSB->SetScrollPos(m_nCellID);
                break;
            case SB_LINERIGHT:
                m_nCellID = pSB->GetScrollPos() + 1;

```



```

        if (m_nCellID > nMax)
            m_nCellID = nMin; // Wrap
        pSB->SetScrollPos(m_nCellID);
        break;
    case SB_PAGELEFT:
        m_nCellID = pSB->GetScrollPos() - (nMax - nMin) / 5;
        if (m_nCellID < nMin)
            m_nCellID = nMin;
        pSB->SetScrollPos(m_nCellID);
        break;
    case SB_PAGERIGHT:
        m_nCellID = pSB->GetScrollPos() + (nMax - nMin) / 5;
        if (m_nCellID > nMax)
            m_nCellID = nMax;
        pSB->SetScrollPos(m_nCellID);
        break;
    }
    if (m_pPS)
    {
        InvalidateRect(&m_rcDraw, FALSE);
    }
    /*
    // Synchronize the sprite in CMapEdView with this selection
    CMapEdView* pMEV = GetDocument()->GetMapEdView();
    CPhasedSprite* pSV = pMEV->GetCapturedSprite();
    if (m_pPS->IsSameDIB(pSV))
    {
        pSV->SetCell(m_nCellID);
        pMEV->RenderAndDrawDirtyList();
    }
    */

    m_strCellInfo.Format("%s (%d/%d)", strName, m_nCellID, nCells);
    UpdateData(FALSE);
}
CFormView::OnHScroll(nSBCode, nPos, pScrollBar);
}

void CMapListView::OnDraw(CDC* pDC)
{
    // CFormView::OnDraw(pDC);
    // OnPrepareDC(pDC); // CScroll changes the viewport origin and mapping
    mode.
    CRect rcDraw(m_rcDraw);
    // pDC->DPTOLP(&rcDraw); // THIS WAS THE KEY!

    if (!m_pPS || (m_nIndex < 0))
        return;

    // If we have a palette, select and realize it.
    CPalette* pPalOld = NULL;
    CMapEdDoc* pDoc = GetDocument();
    CMapEdView* pMEV = pDoc->GetMapEdView();
    CPalette* pPal = (CPalette*)pMEV->GetOSBPalette();
    if (pPal)
    {
        pPalOld = pDC->SelectPalette(pPal, FALSE); //
        bForceBackground = FALSE
    }
}

```

```

        pDC->RealizePalette(); // we realize in response to
WM_QUERYNEWPALETTE
    }

    CPoint ptLT(rcDraw.TopLeft());
    if (gResMan.GetSpriteType(m_nIndex) == SPRITE_TILE)
    {
        int nCells = m_pPS->GetNumCells();
        int w = m_pPS->GetWidth() + 2;
        int h = m_pPS->GetHeight() + 2;
        for (int i=0; i < nCells; i++)
        {
            ptLT.y = rcDraw.top + h * (i/3);
            ptLT.x = rcDraw.left + w * (i%3);
            m_pPS->SetCell(i);
            m_pPS->Draw(pDC, ptLT);
            if (i == m_nCellID)
            {
                CPen pen(PS_SOLID, 1, PALETTERGB(255,0,0));
                pDC->SelectObject(&pen);
                pDC->MoveTo(ptLT);
                pDC->LineTo(ptLT.x+w-3, ptLT.y);
                pDC->LineTo(ptLT.x+w-3, ptLT.y+h-3);
                pDC->LineTo(ptLT.x, ptLT.y+h-3);
                pDC->LineTo(ptLT);
            }
        }
    }
    else
    {
        m_pPS->SetCell(m_nCellID);
        m_pPS->Draw(pDC, ptLT);
        // Draw Earth point
        CSize sE(m_pPS->GetEarth());
        CPen pen(PS_SOLID, 1, PALETTERGB(255,0,255)); // Magenta
        pDC->SelectObject(&pen);
        pDC->MoveTo(ptLT.x, ptLT.y + sE.cy);
        pDC->LineTo(ptLT.x + m_pPS->GetWidth(), ptLT.y + sE.cy);
        pDC->MoveTo(ptLT.x + sE.cx, ptLT.y);
        pDC->LineTo(ptLT.x + sE.cx, ptLT.y + m_pPS->GetHeight());
    }
    // Select old palette if we altered it.
    if (pPalOld)
        pDC->SelectPalette(pPalOld, FALSE);
}

int CMapListView::GetSelectedResName(CString& strName) const
{
    if (m_nIndex >= 0)
        strName = m_lcObject.GetItemText(m_nIndex, 0); // SubItem=0
    return m_nIndex;
}

WORD CMapListView::GetSelectedResType() const
{
    return ((m_nIndex >= 0) ? gResMan.GetSpriteType(m_nIndex) : 0);
}

```

```

void CMapListView::OnLButtonDown(UINT nFlags, CPoint point)
{
    if (!m_pPS || (m_nIndex < 0) || !m_rcDraw.PtInRect(point))
        return;
    ::SetCursor(AfxGetApp()->LoadCursor(IDC_GREP_DOWN));
    if (GetSelectedResType() == SPRITE_TILE)
    {
        int x = m_rcDraw.left;
        int y = m_rcDraw.top;
        int w = m_pPS->GetWidth() + 2;
        int h = m_pPS->GetHeight() + 2;
        int i = ((point.y - m_rcDraw.top)/h) * 3 +
                ((point.x - m_rcDraw.left)/w);
        if (i < m_pPS->GetNumCells())
        {
            m_nCellID = i;
            m_pPS->SetCell(m_nCellID);
            CScrollBar* pSB = (CScrollBar*)GetDlgItem(IDC_SB_CELL_ID);
            OnHScroll(SB_THUMBPOSITION, m_nCellID, pSB);
        }
    }

    CFormView::OnLButtonDown(nFlags, point);
}

void CMapListView::OnMouseMove(UINT nFlags, CPoint point)
{
    if (m_pPS && (m_nIndex >= 0) && m_rcDraw.PtInRect(point))
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_GREP_UP));
    }
    CFormView::OnMouseMove(nFlags, point);
}

// Document Window calls this
void CMapListView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    if (m_lcoObject.GetSafeHwnd()) // Invalidates any calls before
        OnInitialUpdate()
        UpdateList();
}

void CMapListView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    /*
        // switch for input key
        switch (nChar)
        {
        case VK_UP:      pDoc->SendCommand(CMD_MOVE_NORTH); break;
        case VK_LEFT:    pDoc->SendCommand(CMD_MOVE_WEST); break;
        case VK_DOWN:    pDoc->SendCommand(CMD_MOVE_SOUTH); break;
        case VK_RIGHT:   pDoc->SendCommand(CMD_MOVE_EAST); break;
        }
    */
    CFormView::OnKeyDown(nChar, nRepCnt, nFlags);
}

```

```

void CMapListView::SetSelectedItem(const int nIndex)
{
    if (m_nIndex == nIndex)
        return;
    m_lcObject.SetItemState(m_nIndex, 0, LVIF_STATE);
    m_lcObject.SetItemState(nIndex, LVIS_SELECTED | LVIS_FOCUSED,
LVIF_STATE);
    m_nIndex = nIndex;
    UpdateItemImage(m_nIndex);
}

void CMapListView::OnPaletteChanged(CWnd* pFocusWnd)
{
    CFormView::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CMapListView::OnQueryNewPalette()
{
    TRACE("CMapListView::OnQueryNewPalette()\n");
    CMapEdDoc* pDoc = GetDocument();
    CMapEdView* pMEV = pDoc->GetMapEdView();
    if (pMEV->GetOSBPalette())
    {
        CDC* pdc = GetDC();
        CPalette* poldpal = pdc->SelectPalette((CPalette*)pMEV-
>GetOSBPalette(),
FALSE); // foreground
        UINT u = pdc->RealizePalette();
        if (poldpal)
            pdc->SelectPalette(poldpal, FALSE);
        ReleaseDC(pdc);
        if (u)
        {
            // Some colors changed so we need to do a repaint.
            Invalidate(); // Repaint the lot.
            return TRUE; // Say we did something.
        }
    }
    return CFormView::OnQueryNewPalette();
}

```

MapEd\MapListView.h

```

#if
!defined(AFX_MAPLISTVIEW_H__7ECC1E16_9C09_11D1_80E2_288A06C10000__INCLUDED_)
#define AFX_MAPLISTVIEW_H__7ECC1E16_9C09_11D1_80E2_288A06C10000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MapListView.h : header file
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

////////////////////////////////////
// CMapListView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

class CMapEdDoc;
class CPhasedSprite;

class CMapListView : public CFormView
{
protected:
    CMapListView();          // protected constructor used by dynamic
creation
    DECLARE_DYNCREATE(CMapListView)

    enum
    {
        IL_TILE=0,
        IL_STATIC,
        IL_ANIMATED,
        IL_AVATAR
    } OBJECT_IL_INDEX;

// Form Data
public:
    //{AFX_DATA(CMapListView)
    enum { IDD = IDD_DIALOG_VIEW };
    CListCtrl    m_lcObject;
    CString      m_strCellInfo;
    int          m_nCellID;
    //}AFX_DATA

// Attributes
public:
    CMapEdDoc*   GetDocument();
    int          GetSelectedResName(CString& strName) const;
    WORD         GetSelectedResType() const;

// Operations
public:

```

```

        void            UpdateList();
        void            UpdateItemImage(const int nIndex);
        void            SetSelectedItem(const int nIndex);

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CMapListView)
public:
    virtual void OnInitialUpdate();
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual void OnDraw(CDC* pDC);
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
//}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CMapListView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
    void Initialize();
    CImageList          m_iObject;
    int                 m_nIndex;    // Current selected item
    CPhasedSprite*      m_pPS;       // Selected item's image
    CRect               m_rcDraw;    // Draw Region

    // Generated message map functions
    //{{AFX_MSG(CMapListView)
    afx_msg void OnClickListObj(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar);
    afx_msg void OnDblclkListObj(NMHDR* pNMHDR, LRESULT* pResult);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
    afx_msg BOOL OnQueryNewPalette();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BOOL m_bFirst;
};

#ifdef _DEBUG // debug version in MapEdView.cpp
inline CMapEdDoc* CMapListView::GetDocument()
{ return (CMapEdDoc*)m_pDocument; }
#endif

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

```

MapEd\MapListView.h

```
#endif //  
!defined(AFX_MAPLISTVIEW_H__7ECC1E16_9C09_11D1_80E2_288A06C10000__INCLUDED_)
```

```

// PalDlg.cpp : implementation file
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

#include "stdafx.h"
#include "MapEd.h"
#include "PalDlg.h"

#include "MapEdView.h"
#include "../UC2Ani/DIBPal.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

class CDIBPal;

////////////////////////////////////
// CPalDlg dialog

CPalDlg::CPalDlg(CWnd* pParent /*=NULL*/)
: CDialog(CPalDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPalDlg)
    m_bShowIndex = FALSE;
    m_strRGB = _T("");
    m_sbBlue = 0;
    m_sbGreen = 0;
    m_sbRed = 0;
    //}}AFX_DATA_INIT
    m_bCaptured = FALSE;
}

void CPalDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPalDlg)
    DDX_Check(pDX, IDC_CHECK_SHOW_INDEX, m_bShowIndex);
    DDX_Text(pDX, IDC_ST_PAL_RGB, m_strRGB);
    DDX_Scroll(pDX, IDC_SB_PAL_BLUE, m_sbBlue);
    DDX_Scroll(pDX, IDC_SB_PAL_GREEN, m_sbGreen);
    DDX_Scroll(pDX, IDC_SB_PAL_RED, m_sbRed);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPalDlg, CDialog)
    //{{AFX_MSG_MAP(CPalDlg)
    ON_WM_PAINT()

```



```

    ON_BN_CLICKED(IDC_CHECK_SHOW_INDEX, OnCheckShowIndex)
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_MOUSEMOVE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPalDlg message handlers

BOOL CPalDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    GetClientRect(&m_rcPal);

    CWnd* pW = GetDlgItem(IDC_SB_PAL_RED);
    ASSERT(pW);
    CRect rcSB;
    pW->GetClientRect(&rcSB);
    m_rcPal.right -= (rcSB.Width() + 10);

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CPalDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    if (!m_pPal)
        return;

    m_pPal->Draw(&dc, m_rcPal, FALSE, m_bShowIndex);

    // Do not call CDialog::OnPaint() for painting messages
}

void CPalDlg::OnCheckShowIndex()
{
    CButton* pB = (CButton*)GetDlgItem(IDC_CHECK_SHOW_INDEX);
    ASSERT(pB);
    m_bShowIndex = (pB->GetCheck() == 1);
    InvalidateRect(&m_rcPal, FALSE);
}

void CPalDlg::ShowRGB(CPoint point)
{
    HDC hDC = ::GetDC(NULL);
    ClientToScreen(&point);
    COLORREF crRGB = ::GetPixel(hDC, point.x, point.y);
    ::ReleaseDC(NULL, hDC);

    char buf[64];
    char name[30] = "";
    int r = GetRValue(crRGB);
    int g = GetGValue(crRGB);

```

```

    int b = GetBValue(crRGB);
    int i = -1;
    if (m_pPal)
        i = (BYTE)m_pPal->GetNearestPaletteIndex(crRGB);

    if (r==0 && g==0 && b==0) wsprintf(name,
"%s", "Black");
    else if (r==128 && g==0 && b==0) wsprintf(name,
"%s", "Dark Red");
    else if (r==0 && g==128 && b==0) wsprintf(name, "%s", "Dark
Green");
    else if (r==128 && g==128 && b==0) wsprintf(name, "%s",
"Dark Yellow");
    else if (r==0 && g==0 && b==128) wsprintf(name, "%s",
"Dark Blue");
    else if (r==128 && g==0 && b==128) wsprintf(name,
"%s", "Dark Magenta");
    else if (r==0 && g==128 && b==128) wsprintf(name, "%s", "Dark
Cyan");
    else if (r==192 && g==192 && b==192) wsprintf(name, "%s",
"Light Gray");
    else if (r==192 && g==220 && b==192) wsprintf(name, "%s",
"Money Green");
    else if (r==166 && g==202 && b==240) wsprintf(name, "%s",
"Sky Blue");
    else if (r==255 && g==251 && b==240) wsprintf(name, "%s",
"Cream");
    else if (r==160 && g==160 && b==164) wsprintf(name, "%s",
"Medium Gray");
    else if (r==128 && g==128 && b==128) wsprintf(name, "%s",
"Dark Gray");
    else if (r==255 && g==0 && b==0) wsprintf(name,
"%s", "Red");
    else if (r==0 && g==255 && b==0) wsprintf(name, "%s",
"Green");
    else if (r==255 && g==255 && b==0) wsprintf(name, "%s",
"Yellow");
    else if (r==0 && g==0 && b==255) wsprintf(name, "%s",
"Blue");
    else if (r==255 && g==0 && b==255) wsprintf(name,
"%s", "Magenta");
    else if (r==0 && g==255 && b==255) wsprintf(name, "%s", "Cyan");
    else if (r==255 && g==255 && b==255) wsprintf(name, "%s",
"White");

    wsprintf(buf, "%d (%03d,%03d,%03d) %s", i, r, g, b, name);
    SetWindowText(buf);
}

void CPalDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
    ::SetCursor(AfxGetApp()->LoadCursor(IDC_PIXEL));
    SetCapture();
    m_bCaptured = TRUE;
    ShowRGB(point);
    CDialog::OnLButtonDown(nFlags, point);
}

```

```
void CPalDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
    if (m_bCaptured)
    {
        ReleaseCapture();
        m_bCaptured = FALSE;
        // SetWindowText("SysPal"); // Restore title.
    }
    CDialog::OnLButtonUp(nFlags, point);
}

void CPalDlg::OnMouseMove(UINT nFlags, CPoint point)
{
    if (m_bCaptured)
    {
        ::SetCursor(AfxGetApp()->LoadCursor(IDC_PIXEL));
        ShowRGB(point);
    }
    CDialog::OnMouseMove(nFlags, point);
}
```

```

#if !defined(AFX_PALDLG_H__E036BF42_A18D_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_PALDLG_H__E036BF42_A18D_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PalDlg.h : header file
//
//=====
//      (C) Programmed by Kim, Feb 1998
//      Information Technology Institute
//      UNICHAT INC
//=====

class CDIBPal;
////////////////////////////////////
// CPalDlg dialog

class CPalDlg : public CDialog
{
// Construction
public:
    CPalDlg(CWnd* pParent = NULL);    // standard constructor

    void SetPalette(CDIBPal* pPal)    { m_pPal = pPal; }

// Dialog Data
    //{AFX_DATA(CPalDlg)
    enum { IDD = IDD_PALETTE };
    BOOL m_bShowIndex;
    CString m_strRGB;
    int m_sbBlue;
    int m_sbGreen;
    int m_sbRed;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CPalDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    void ShowRGB(CPoint point);

    CDIBPal* m_pPal;
    CRect m_rcPal;
    BOOL m_bCaptured;

    // Generated message map functions
    //{AFX_MSG(CPalDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg void OnCheckShowIndex();

```

MapEd\PalDlg.h

```
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
afx_msg void OnMouseMove(UINT nFlags, CPoint point);
//{{AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_PALDLG_H__E036BF42_A18D_11D1_80E2_080009B9F339__INCLUDED_)
```

=====

MICROSOFT FOUNDATION CLASS LIBRARY : MapEd

=====

AppWizard has created this MapEd application for you. This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that make up your MapEd application.

MapEd.h

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CMapEdApp application class.

MapEd.cpp

This is the main application source file that contains the application class CMapEdApp.

MapEd.rc

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

res\MapEd.ico

This is an icon file, which is used as the application's icon. This icon is included by the main resource file MapEd.rc.

res\MapEd.rc2

This file contains resources that are not edited by Microsoft Developer Studio. You should place all resources not editable by the resource editor in this file.

MapEd.clw

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

////////////////////////////////////

For the main frame window:

MainFrm.h, MainFrm.cpp

These files contain the frame class CMainFrame, which is derived from CFrameWnd and controls all SDI frame features.

res\Toolbar.bmp

This bitmap file is used to create tiled images for the toolbar. The initial toolbar and status bar are constructed in the CMainFrame class. Edit this toolbar bitmap along with the array in MainFrm.cpp to add more toolbar buttons.

////////////////////////////////////////////////////////////////

AppWizard creates one document type and one view:

MapEdDoc.h, MapEdDoc.cpp - the document

These files contain your CMapEdDoc class. Edit these files to add your special document data and to implement file saving and loading (via CMapEdDoc::Serialize).

MapEdView.h, MapEdView.cpp - the view of the document

These files contain your CMapEdView class. CMapEdView objects are used to view CMapEdDoc objects.

////////////////////////////////////////////////////////////////

Other standard files:

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header (PCH) file named MapEd.pch and a precompiled types file named StdAfx.obj.

Resource.h

This is the standard header file, which defines new resource IDs. Microsoft Developer Studio reads and updates this file.

////////////////////////////////////////////////////////////////

Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you should add to or customize.

If your application uses MFC in a shared DLL, and your application is in a language other than the operating system's current language, you will need to copy the corresponding localized resources MFC40XXX.DLL from the Microsoft Visual C++ CD-ROM onto the system or system32 directory, and rename it to be MFCLOC.DLL. ("XXX" stands for the language abbreviation. For example, MFC40DEU.DLL contains resources translated to German.) If you don't do this, some of the UI elements of your application will remain in the language of the operating system.

////////////////////////////////////////////////////////////////

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by MapEd.rc
//
#define IDD_ABOUTBOX 100
#define DIB_GRID_64X32 101
#define ID_SPRITE_TYPE 104
#define ID_ACTOR_ELEVATION 105
#define IDB_SPLASH 106
#define DIB_ROOMKEY 107
#define IDR_MAINFRAME 128
#define IDR_MAPEDTYPE 129
#define IDS_ENTER_MSPT 129
#define IDD_DIALOG_VIEW 130
#define IDS_ENTER_ELEVATION 131
#define IDS_ENTER_ACTOR_ELEVATION 132
#define IDS_ENTER_TILE_ELEVS 133
#define IDR_MENU_TILE 134
#define IDS_ENTER_STAGE_TITLE 135
#define IDS_ENTER_STAGE_MUSIC 136
#define IDS_ENTER_STAGE_LINK 137
#define IDB_IL_OBJECT 138
#define IDD_ENVIRONMENT 139
#define IDC_PIXEL 140
#define IDD_CLOSE 141
#define IDD_GETINT 142
#define IDD_GETSTRING 143
#define IDD_PALETTE 144
#define IDR_MENU_GRID 145
#define IDC_GREP_UP 159
#define IDC_GREP_DOWN 160
#define IDC_LIST_OBJ 1000
#define IDC_SB_PAL_RED 1000
#define IDC_ST_CELL_INFO 1001
#define IDC_SB_PAL_GREEN 1001
#define IDC_SB_CELL_ID 1002
#define IDC_SB_PAL_BLUE 1002
#define IDC_ST_PAL_RGB 1003
#define IDC_EDIT_SCREEN_W 1003
#define IDC_CHECK_SHOW_INDEX 1004
#define IDC_EDIT_SCREEN_H 1004
#define IDC_EDIT_TILE_W 1005
#define IDC_EDIT_TILE_H 1006
#define IDC_EDIT_DATA_PATH 1007
#define IDC_EDIT_PALFILE 1008
#define IDC_SPIN1 1011
#define IDC_ST_CURRENT_INT 1013
#define IDC_EDIT_NEW_INT 1014
#define IDC_EDIT_STRING 1015
#define IDS_RIT_NO_BEHAVIORS 1058
#define IDS_RIT_NO_SPRITES 1059
#define IDS_RIT_NO WAVES 1060
#define IDS_RIT_NO_MIDIS 1061
#define IDS_RIT_NO_STAGES 1062
#define IDS_RIT_NO_SERVERIPS 1063
#define ID_FILE_LOADBKGND 32771
#define ID_FILE_LOADSPRITE 32772

```



```

#define ID_VIEW_GRID 32776
#define ID_FILE_LOAD_RIT 32777
#define ID_FILE_LOAD_STAGE 32779
#define ID_VIEW_EARTH 32783
#define ID_FILE_LOAD_MAP 32784
#define ID_FILE_SAVE_STAGE 32785
#define ID_FILE_SAVE_MAP 32786
#define ID_VIEW_PALETTE 32787
#define ID_OPTIONS_ENV 32792
#define ID_EDIT_MODE 32794
#define ID_SPRITE_ELEVATION 32796
#define ID_SPRITE_FLIP 32797
#define ID_SPRITE_VERTICAL 32798
#define ID_SPRITE_OPACITY100 32799
#define ID_SPRITE_OPACITY50 32800
#define ID_SPRITE_OPACITY25 32801
#define ID_SPRITE_OPACITY12 32802
#define ID_SPRITE_DELETE 32803
#define ID_SPRITE_FLIPVERT 32804
#define ID_VIEW_MAP_ONLY 32805
#define ID_VIEW_GRAPH 32808
#define ID_EDIT_TILE_COORD 32810
#define ID_VIEW_PAUSE 32812
#define ID_MENU_ANIMATION 32814
#define ID_MENU_ANI_REPEAT 32815
#define ID_MENU_ANI_FADE 32816
#define ID_MENU_NOANI 32817
#define ID_SPRITE_OPACITY75 32818
#define ID_MENU_ANI_RANDOM 32819
#define ID_MENU_STAGE_TITLE 32820
#define ID_EDIT_STAND 32822
#define ID_MENU_DA_FR 32825
#define ID_MENU_DA_FL 32826
#define ID_MENU_DA_BL 32827
#define ID_MENU_DA_BR 32829
#define ID_EDIT_CREATE_MODE 32830
#define ID_MENU_RECALC_DA 32832
#define ID_MENU_DA_OPEN 32833
#define ID_MENU_DA_CLOSED 32834
#define ID_MENU_SYNC_EA 32835
#define ID_VIEW_NOELEV 32836
#define ID_MENU_STAGE_MUSIC 32837
#define ID_VIEW_EA_GRAPH 32838
#define ID_MENU_ELEVATION 32840
#define ID_MENU_EXIT_POINT 32841
#define ID_MENU_ENTRY_POINT 32842

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS 1
#define _APS_NEXT_RESOURCE_VALUE 145
#define _APS_NEXT_COMMAND_VALUE 32843
#define _APS_NEXT_CONTROL_VALUE 1016

```

```
#define _APS_NEXT_SYMED_VALUE    108
#endif
#endif
```

MapEd\StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//      MapEd.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information
```

```
#include "stdafx.h"
```

```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#ifdef !defined(AFX_STDAFX_H__7ECC1E08_9C09_11D1_80E2_288A06C10000__INCLUDED_)
#define AFX_STDAFX_H__7ECC1E08_9C09_11D1_80E2_288A06C10000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>             // MFC core and standard components
#include <afxext.h>             // MFC extensions
#include <afxdisp.h>           // MFC OLE automation classes
#include <afxinet.h>
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include "../UC2Messages.h"

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
#endif(AFX_STDAFX_H__7ECC1E08_9C09_11D1_80E2_288A06C10000__INCLUDED_)

```

```

//-----
//  CBaseSocket class
//
//  (C) Programmed by Kim
//  UNICHAT Media Lab
//  Information Technology Institute
//  UNICHAT
//
//  Copyright (C) 1996 Microsoft Corporation
//  All rights reserved.
//
//  Modified and trouble-shooted for MFC
//  ::CreateThread ----> CWinThread* AfxBeginThread
//
//  Modified by Hojun Park. Feb 18
//  1. Add FQueryListAllUsers(), FQueryListChannels() to this class.
//-----
#include "stdafx.h"
#include "PXMessages.h"
#include "BaseSock.h"

//DWORD __stdcall DwSocketThreadProc(PVOID pvData);

CBaseSocket::CBaseSocket()
{
    m_pics                = NULL;
    m_pFactory             = NULL;
    m_pThread             = NULL;
    m_hThread             = NULL;
    m_bCanceled           = FALSE;
    m_nPICSRef            = 0;
    m_bQueryOK            = TRUE;
    m_LastQType           = Q_LIST_USERS;
    m_iItem               = 0;
    m_hNotify             = NULL;
}

CBaseSocket::~CBaseSocket()
{
    CleanUp();
}

//  Close the socket and kill the worker thread.
void CBaseSocket::CleanUp()
{
    if (!m_bCanceled)
        FCloseChatSocket();

    if (m_pFactory)
    {
        m_pFactory->Release();
    }
    if (m_pics)
    {
        m_pics->Release();
    }
}

```

```

        WaitForMsgThread();
    }

    // We need to make sure to wait for the thread to exit, so we
    // can be sure all the resources have been cleaned up.
    void CBaseSocket::WaitForMsgThread()
    {
        if (m_hThread)
        {
            TRACE("CBaseSocket - ::WaitForSingleObject(0x%lx, 1000L);\n",
m_hThread);

            DWORD dwRc = ::WaitForSingleObject(m_hThread, 1000L);
            switch (dwRc)
            {
            case WAIT_OBJECT_0:
                TRACE0("Thread: WAIT_OBJECT_0\n");
                break;
            case WAIT_ABANDONED:
                TRACE0("Thread: WAIT_ABANDONED\n");
                break;
            case WAIT_TIMEOUT:
                TRACE0("Thread Hung! Deleting...\n");
                delete m_pThread;
                break;
            case WAIT_FAILED:
                TRACE("Thread: WAIT_FAILED ");
                break;
            default:
                TRACE0("Thread Hung!\n");
                break;
            }
            //::CloseHandle(m_hThread);
            m_pThread = NULL; // CWinThread auto-deleted
            m_hThread = NULL;
        }
    }

    // Log off and close the socket.
    BOOL CBaseSocket::FCloseChatSocket()
    {
        if (m_pics)
        {
            m_pics->HrLogOff();

            // After a successful log off,
            // the IChatSocket::HrWaitForMsg method will return a
            CS_E_QUEUEEMPTY message,
            // You should then terminate your message loop.
            WaitForMsgThread(); // Soomin Kim

            m_pics->HrCloseSocket(); // A thread in ChatSock.dll exits.
            (0x80004601C)
            m_pics->Release();

            if (m_nPICSRef)
            {

```

```

        m_pics->Release();
        m_nPICSRef--;
    }

    m_pics = NULL;
}

return TRUE;
}

PICS CBaseSocket::PChatSocket()
{
    ASSERT(m_pics);

    if (m_pics)
        m_pics->AddRef();
    return m_pics;
}

// Initialize the object. We create the socket factory here, so we can
// allow disconnecting during the process of connecting; a connection attempt
// can take a long time, so if the caller wants to cancel the process,
// it can call HrDisconnect() in another thread.
BOOL CBaseSocket::FInit()
{
    if (m_pics)
    {
        // we already have it.
        return FALSE;
    }

    if (m_pFactory)
    {
        // We already have it, just Release() it.
        m_pFactory->Release();
    }

    TRACE0("CBaseSocket: ::HrCreateChatSocketFactory(IID_CHATSOCKVER1,
&m_pFactory)\n");

    HRESULT hr = ::HrCreateChatSocketFactory(IID_CHATSOCKVER1,
&m_pFactory);
    if (FAILED(hr))
    {
        // If we failed to get the interface IChatSocketFactory.
        FOnSocketError(hr); // virtual function call
        m_pFactory = NULL;
        return FALSE;
    }
    return TRUE;
}

// Connecting to a chat server via ChatSock is a 2-step process.
// Since we don't know if the chat server is a MIC or IRC server,
// we call the socket factory and let it sort out the protocol issues.
// The socket factory will create a socket.
// Once we have the socket, we can then connect and log into the server.

```

```

BOOL CBaseSocket::FConnect(PEC_CONNINFO pcInfo, HWND hNotifyWnd)
{
    HRESULT          hr = NOERROR;
    BOOL             fDoneBak = FALSE; // Have we tried the backup nick?
    BOOL             fLoop = TRUE;
    TCHAR            cBackupIDSuffix = '0';

    ASSERT(pcInfo && m_pFactory);

    if (FConnected())
        return TRUE;

    // If we have a window to which we notify event, We notify the event to
    the window.
    // CMD_CONNECT_CONNECTING is a predefined message defined in
    PXMessage.h.
    if (hNotifyWnd)
    {
        m_hNotify = hNotifyWnd;
        ::PostMessage(hNotifyWnd, CMD_CONNECT_CONNECTING, 0, 0);
    }

    m_bCanceled = FALSE;
    TRACE("CBaseSocket: m_pFactory->HrMakeSocket(%s, %lx)\n", pcInfo->szServer, &m_pics);

    hr = m_pFactory->HrMakeSocket(pcInfo->szServer, &m_pics);
    if (FAILED(hr))
        goto LReturn;

    if (m_pics->HrIsMicSocket() != NOERROR)
    {
        // whether we connect to MIC Server. If NO.
        ::MessageBox(NULL, "This is not an MIC socket!\n", "CBaseSocket",
        MB_OK);
        goto LReturn;
    }

    // If we have a window to which we notify event, We notify the event to
    the window.
    // CMD_CONNECT_LOGIN is a predefined message defined in PXMessage.h.
    if (hNotifyWnd)
        ::PostMessage(hNotifyWnd, CMD_CONNECT_LOGIN, 0, 0);

    CS_CONNINFO cInfo;
    ::ZeroMemory(&cInfo, sizeof(CS_CONNINFO));
    cInfo.dwcb = sizeof(CS_CONNINFO);
    cInfo.bType = pcInfo->fAuthenticate ? CS_CONNECT_AUTHENTICATE :
    CS_CONNECT_ANONYMOUS;
    cInfo.pvNick = (PVOID)pcInfo->szNick;
    cInfo.pvUser = (PVOID)pcInfo->szUserName;
    cInfo.pvPass = (PVOID)pcInfo->szPass;

    // Loop till we have a result on the login
    while (fLoop)
    {
        hr = m_pics->HrLoginA(&cInfo);
    }
}

```



```

        if (FAILED(hr))
        {
            goto LReturn;
        }

        // Now check the Wait Q for acknowledgement
        PCS_MSGBASE pcsMsg;
        hr = m_pics->HrWaitTillMsgType(CSMMSG_TYPE_LOGIN, &pcsMsg, pcInfo-
>dwTimeOut);
        if (FAILED(hr))
        {
            // Did we log in or get an error?
            if (hr == CS_E_ILLEGALUSER)
            {
                FOnSocketError(hr);
                continue;
            }

            if ((hr == CS_E_ALIASINUSE) && !fDoneBak && pcInfo-
>szNickBak)
            {
                int n = lstrlen(pcInfo->szNickBak);
                pcInfo->szNickBak[n-1] = cBackupIDSuffix;
                cInfo.pvNick = (PVOID)pcInfo->szNickBak;
                if (cBackupIDSuffix++ >= '9')
                {
                    fDoneBak = TRUE;
                }

                if (hNotifyWnd)
                {
                    ::PostMessage(hNotifyWnd, CMD_CONNECT_BACKUPID,
0, 0);
                }
                continue;
            }
            goto LReturn;
        }

        FOnLogin();          // call the virtual function
        fLoop = FALSE;
        ::HrFreeMsg(pcsMsg);
    }

LReturn:
    // If successful, save the socket
    if (SUCCEEDED(hr))
    {
        // Start a thread to read messages from the message queue.
        if (!FStartMessageThread())
        {
            FOnSocketError(E_OUTOFMEMORY);          // virtual function

            CleanUp();
            return FALSE;
        }
    }
}

```

```

        return TRUE;
    }

    // Failure
    // On Cancel, m_pics->HrCloseSocket() may already been called by
    FCloseChatSocket().
    if (!m_bCanceled && m_pics)
    {
        TRACE0("CBaseSocket::FConnect: m_pics->HrCloseSocket()\n");
        m_pics->HrCloseSocket();
        TRACE0("CBaseSocket::FConnect: m_pics->Release()\n");
        m_pics->Release();
        m_pics = NULL;
    }

    // FOnSocketError(hr);      // virtual function call      // Doesn't work
    for OnCancel...
    // TRACE0("CBaseSocket::FConnect returned FALSE\n");
    return FALSE;
}

// Disconnect from the server.
// Cancels any in-progress connection attempts or other blocking operations.
BOOL CBaseSocket::FDisconnect()
{
    // As long as m_pFactory is alive, HrCancelMakeSocket() will have
    // the same effect as calling HrCloseSocket() on a ChatSocket.
    m_bCanceled = TRUE;
    FCloseChatSocket();
    if (m_pFactory)
    {
        TRACE0("CBaseSocket: m_pFactory->HrCancelMakeSocket()\n");
        HRESULT hr = m_pFactory->HrCancelMakeSocket();
        m_pFactory->Release();
        m_pFactory = NULL;      // to prevent calling m_pFactory-
        >HrCancelMakeSocket() again
        if (FAILED(hr))
        {
            FOnSocketError(hr);      // virtual function call
            return FALSE;
        }
    }

    // m_pThread = NULL;
    // m_hThread = FALSE;
    return TRUE;
}

// Does this server support anonymous logins?
BOOL CBaseSocket::FCanAnonymous()
{
    // Grab security details.
    PCS_SECURITY pcsSecurity;
    HRESULT hr = m_pics->HrGetSecurityInfo(&pcsSecurity);
    if (FAILED(hr))
    {
        return TRUE;      // failed to get the correct answer. just
        assume that anon.
    }
}

```

```

    }
    return pcsSecurity->fAnonAllowed;
}

// Uses m_pics->HrCreateChannelA() to create a channel;
// If a channel with the specified name already exists,
// This function simply joins that channel.
BOOL CBaseSocket::FCreateJoinChannel(PEC_CHANNELINFO pChanInfo)
{
    ASSERT(pChanInfo);
    ASSERT(m_pics);

    CS_CINFO    cInfo; // defined in csface.h
    ::FillMemory(&cInfo, sizeof(CS_CINFO), 0);

    cInfo.dwc    = sizeof(CS_CINFO);
    cInfo.dwType = pChanInfo->dwType;
    cInfo.dwFlags = pChanInfo->dwFlags;
    // if channel exists, join it, else create a new one.
    cInfo.bCreateFlags = CS_CHANNEL_CREATE_JOIN;
    cInfo.pvChannelName = (PVOID)pChanInfo->szName;
    cInfo.pvTopic       = (PVOID)pChanInfo->szTopic;
    cInfo.pvPassword    = pChanInfo->szPass;
    cInfo.dwcUserMax    = pChanInfo->cUsersMax;

    HRESULT hr = m_pics->HrCreateChannelA(&cInfo);
    if (FAILED(hr))
    {
        FOnSocketError(hr);    // virtual function call
        return FALSE;
    }
    return TRUE;
}

BOOL CBaseSocket::FConnected()
{
    return (m_pics && (NOERROR == m_pics->HrIsConnected()));
}

BOOL CBaseSocket::ChangeNick(LPCSTR nick)
{
    ASSERT(m_pics);

    HRESULT hr = m_pics->HrChangeNickA((char*)nick);
    if (FAILED(hr))
    {
        FOnSocketError(hr);    // virtual function call
        return FALSE;
    }
    return TRUE;
}

// Waits for a message to arrive on the message queue.
// Calling FCloseChatSocket on the socket will cause this method to return
FALSE immediately.
// Dispatches the received message using the overrideable virtual methods
of CBaseSocket.

```

```

BOOL CBaseSocket::FWaitForMessage()
{
    if (!m_pics)
        return FALSE;

    m_pics->AddRef(); // increase the ref count so that
we can be sure that
    TRACE0("m_pics->AddRef();\n"); // the socket object doesn't go
away until this function ends...
    m_nPICSRef++;

    PCS_MSGBASE pcsMsg;
    while (SUCCEEDED(m_pics->HrWaitForMsg(&pcsMsg, INFINITE)))
    {
        TRACE("=> CBaseSocket::FWaitForMessage\n", pcsMsg->csMsgType);
        switch (pcsMsg->csMsgType) // BYTE
        {
            case CSMSG_TYPE_ERROR:
            {
                TRACE("CSMSG_TYPE_ERROR\n");
                PCS_ERROR pErr = MSGBASE_TO_MSG(pcsMsg, PCS_ERROR);
                FOnSocketError(pErr->hr); // virtual function

                if (pErr->hr == CS_E_NOMATCHES)
                {
                    SetQueryOK();
                    ::SendMessage(m_hNotify,
CMD_QUERY_CHANNELS_END, (LPARAM)0, (LPARAM)0);
                    TRACE("No Matches!\n");
                }
                break;

            case CSMSG_TYPE_ADDCHANNEL:
            {
                PCS_MSGCHANNEL pMsgCh = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGCHANNEL);
                FOnAddChannel(pMsgCh); // virtual function
                break;

            case CSMSG_TYPE_PRIVATEMSG:
            {
                PCS_MSGPRIVATE pMsgPrivate = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGPRIVATE);
                FOnPrivateMsg(pMsgPrivate); // virtual function
                break;

            case CSMSG_TYPE_QUERYDATA:
            {
                TRACE("CBaseSocket::FWaitForMessage--
>CSMSG_TYPE_QUERYDATA\n");

```

```

        PCS_PROPERTY      pcsProp = MSGBASE_TO_MSG(pcsMsg,
PCS_PROPERTY);
        FParseQueryData(pcsProp);      // virtual function
call
    }
    break;

    case CSMSG_TYPE_INVITE:
    {
        PCS_MSGINVITE      pcsInvite = MSGBASE_TO_MSG(pcsMsg,
PCS_MSGINVITE);
        FOnInvite(pcsInvite);      // virtual function
call
    }
    break;

    default:
        FUnknownMessage(pcsMsg);      // virtual function call
        break;
    }
    ::HrFreeMsg(pcsMsg);
}
if (m_pics)
{
    m_pics->Release();
    m_nPICSRef--;
    TRACE0("m_pics->Release();\n");      // the socket object doesn't
go away until this function ends...
//    m_pics = NULL;
}
TRACE0("End of CBaseSocket::FWaitForMessage()\n");
return TRUE;
}

// Performs default parsing on query data messages.
// If this method is not overridden, it will call FOnPropString and
FOnPropBuffer
// as appropriate for the query data.
BOOL CBaseSocket::FParseQueryData(PCS_PROPERTY pcsProp)
{
    BOOL fRet = TRUE;
    ASSERT(pcsProp);
    if (!pcsProp->picsProperty)
    {
        // Null Property pointers indicate that
        // this is the last record in a data set returned by a query.
        SetQueryOK();
        return TRUE;
    }

    switch (GetLastQueryType())
    {
    case Q_LIST_CHANNELS:
    {
        TRACE("Q_LIST_CHANNELS\n");
        ::SendMessage(m_hNotify, CMD_QUERY_CHANNELS, 0,
(LPARAM)pcsProp->picsProperty);
    }
    }
}

```

```

        m_iItem++;
        if (pcsProp->fLastRecord)
        {
            TRACE("Q_LIST_CHANNELS-->fLastRecorded\n");
            SetQueryOK();
            ::SendMessage(m_hNotify,
                CMD_QUERY_CHANNELS_END, (WPARAM)m_iItem,
(LPARAM) 0);
        }
    }
    break;
case Q_LIST_USERS:
case Q_CHANNEL_MEMBERS: // CMemberDlg inquires
    {
        TRACE("Q_CHANNEL_MEMBERS\n");
        ::SendMessage(m_hNotify, CMD_QUERY_MEMBERS, 0,
            (LPARAM)pcsProp->picsProperty);
        m_iItem++;
        if (pcsProp->fLastRecord)
        {
            TRACE("Q_CHANNEL_MEMBERS-->fLastRecord\n");
            SetQueryOK();
            ::SendMessage(m_hNotify, CMD_QUERY_MEMBERS_END,
                (WPARAM)m_iItem, (LPARAM)0);
        }
    }
    break;
case Q_GET_REALNAME: // CSINDEX_PROP_REALNAME_NAME
    {
        CS_PROPDATA cspd;
        if (FAILED(pcsProp->picsProperty->HrGetProperty(&cspd,
CSINDEX_PROP_REALNAME_REALNAME)))
            return FALSE;
        CHAR* szName = (CHAR*)cspd.pbData;
        ::SendMessage(m_hNotify,
            CSMSG_CMD_QUERYDATA, 0, (LPARAM)szName);
        TRACE("Q_GET_REALNAME =>(%s)\n", szName);
        SetQueryOK();
    }
    break;
default:
    break;
}
return fRet;
}

BOOL CBaseSocket::FStartMessageThread()
{
    WaitForMsgThread();

    m_pThread = AfxBeginThread(SocketThreadProc, (LPVOID)this);
    ASSERT(m_pThread);
    m_hThread = m_pThread->m_hThread; // Save the handle
    return (m_pThread != NULL);
}

```

```

// this function must be static and prototyped like this.
UINT CBaseSocket::SocketThreadProc(LPVOID pvData)
{
    ASSERT(pvData);
    CBaseSocket* pbasesocket = (CBaseSocket*)pvData;
    return pbasesocket->FWaitForMessage();
}

////////////////////////////////////
// Operations via Chat Socket
BOOL CBaseSocket::FSendPrivAnsiText(char* szNickTo, char* szText)
{
    ASSERT(m_pics);

    CS_PRIVMSGINFO msg;
    msg.dwcb = sizeof(CS_PRIVMSGINFO);
    msg.dwUserID = 0; // 0 if not available
    msg.pvNickTo = szNickTo; // name of the user to send
    this msg to
    msg.fData = FALSE; //
    msg.pbData = (BYTE*)szText; // data to send
    msg.dwcbData = 0; // 0 means null terminated
    string
    HRESULT hr = m_pics->HrSendPrivMsgA(&msg);
    if (FAILED(hr))
    {
        FOnSocketError(hr); // virtual function call
        return FALSE;
    }
    return TRUE;
}

BOOL CBaseSocket::FSendPrivData(char* szNickTo, BYTE* pbData, DWORD dwcb)
{
    ASSERT(m_pics);

    CS_PRIVMSGINFO msg;
    msg.dwcb = sizeof(CS_PRIVMSGINFO);
    msg.dwUserID = 0; // 0 if not available
    msg.pvNickTo = szNickTo; // name of the user to send this
    msg to
    msg.fData = TRUE; //
    msg.pbData = pbData; // data to send
    msg.dwcbData = dwcb; // 0 means null terminated
    string
    HRESULT hr = m_pics->HrSendPrivMsgA(&msg);
    if (FAILED(hr))
    {
        FOnSocketError(hr); // virtual function call
        return FALSE;
    }
    return TRUE;
}

BOOL CBaseSocket::FQueryListAllUsers(char* szFind)
{

```

```

    TRACE("CBaseSocket::FQueryListAllUsers\n");

    ASSERT(m_pics);

    if (!IsQueryOK())
    {
        ::MessageBox(NULL, "Receiving Data.... QueryListAllUsers",
"Socket Report",
        MB_OK | MB_ICONEXCLAMATION);
        return FALSE;
    }

    HRESULT hr = m_pics->HrListAllUsersA(szFind, CSROP_QUERY_CONTAINS);
    if (FAILED(hr))
    {
        FOnSocketError(hr);    // virtual function call
        return FALSE;
    }
    SetLastQueryType(Q_LIST_USERS);
    return TRUE;
}

// Lists all channels that meet the following criteria:
// a) the channel name must contain szChannel, and
// b) the channel must have a number of members within the specified
range.
BOOL CBaseSocket::FQueryListChannels(char* szChannel, DWORD dwcMin, DWORD
dwcMax)
{
    TRACE("CBaseSocket::FQueryListChannels\n");

    // Default
    // char* szChannel=NULL, DWORD dwcMin=1, DWORD dwcMax=1000
    ASSERT(m_pics);

    if (!IsQueryOK())
    {
        TRACE("IsQueryOK() is FALSE\n");

        ::MessageBox(NULL, "Receiving Data... QueryListChannels", "Socket
Report", MB_OK | MB_ICONEXCLAMATION);
        return FALSE;
    }

    HRESULT hr = m_pics->HrListAllChannelsA(dwcMin, dwcMax, szChannel,
CSROP_QUERY_CONTAINS);
    if (FAILED(hr))
    {
        FOnSocketError(hr);    // virtual function call
        return FALSE;
    }
    SetLastQueryType(Q_LIST_CHANNELS);
    return TRUE;
}

void CBaseSocket::SetLastQueryType(const int nType)
{

```



```
TRACE("CBaseSocket::SetLastQueryType\n");

m_LastQType = nType;
SetQueryOK(FALSE);
m_iItem = 0;
}

////////////////////////////////////
/////
// Debugging
```

```

//-----
//  CBaseSocket class
//
//  Copyright (C) 1996 Microsoft Corporation
//  All rights reserved.
//
//  Modification for MFC
//  (C) Programmed by Kim
//
//  Information Technology Institue
//  UNICHAT INC
//  Modified by Hojun Park. Feb 18
//  1. Add FQueryListAllUsers(), FQueryListChannels() to this class.
//-----
#ifndef _BASESOCK_H
#define _BASESOCK_H

#include <micmsg.h>           // MIC protocol header
#include <csface.h>          // ChatSock header
#include <cserror.h>         // ChatSock errors
////////////////////////////////////

// class CBaseSocket

// Pointer to this structure is passed to CBaseSocket::HrConnect
typedef struct conninfo
{
    char* szServer;           // server to connect to
    char* szNick;             // Nickname to use
    char* szNickBak;         // A backup nick - optional
    char* szUserName;        // user name to use
    char* szPass;            // if szUserName is provided, so must
szPass
    BOOL fAuthenticate;      // should the connection be authenticated?
    DWORD dwTimeOut;
} EC_CONNINFO, *PEC_CONNINFO;

// Pointer to this structure is passed to CBaseSocket::HrCreateJoinChannel
typedef struct channelinfo
{
    CHAR* szName;             // name of the channel to join
    CHAR* szTopic;            // if channel did not exist, and was therefore
                                // created for you, set the topic
to this value
    CHAR* szPass;             // password on this channel
    DWORD cUsersMax;          // how many users at max...0 if default
    DWORD dwType;             // channel type
    DWORD dwFlags;            // channel flags.
} EC_CHANNELINFO, *PEC_CHANNELINFO;

// CBaseSocket
// CBaseSocket is a wrapper around IChatSocket. It provides message-handling
code
// and wrappers around some IChatSocket functionality. If you need more
// functionality from ChatSock, use CBaseSocket::PSocket() to obtain a socket
// interface and then call ChatSock directly.
// Note: if you intend to keep that pointer around in other data structures,

```

```
// you should AddRef() it and then Release() it when that particular data
// structure
// goes away. This will make your cleanup logic a lot more robust.
```

```
class CBaseSocket : public CObject
{
public:
    BOOL ChangeNick(LPCSTR nick);
    CBaseSocket();
    virtual ~CBaseSocket();

    enum
    {
        Q_LIST_USERS,
        Q_LIST_CHANNELS,
        Q_CHANNEL_MEMBERS,
        Q_GET_REALNAME
    } QUERY_TYPE;

    BOOL IsQueryOK() const { return
m_bQueryOK; }
    void SetQueryOK(const BOOL bQOK=TRUE) { m_bQueryOK =
bQOK; }
    int GetLastQueryType() const {
return m_LastQType; }
    void SetLastQueryType(const int nType);

    BOOL FInit();
    BOOL FConnect(PEC_CONNINFO pcInfo, HWND hNotifyWnd=NULL);
    BOOL FDisconnect();
    BOOL FCloseChatSocket();

    BOOL FCreateJoinChannel(PEC_CHANNELINFO pChanInfo);

    BOOL FCanAnonymous();
    BOOL FConnected();
    PICS PChatSocket();

    BOOL FSendPrivAnsiText(char* szNickTo, char* szText);
    BOOL FSendPrivData(char* szNickTo, BYTE* pbData, DWORD
dwcb);
    BOOL FQueryListChannels(char* szChannel=NULL, DWORD
dwcMin=1, DWORD dwcMax=1000);
    BOOL FQueryListAllUsers(char* szFind=NULL);

    // Overrideables
    virtual BOOL FOnLogin() {
return TRUE; }
    virtual BOOL FOnSocketError(HRESULT hr) { return
TRUE; }
    virtual BOOL FOnAddChannel(PCS_MSGCHANNEL pMsg) { return TRUE; }
    virtual BOOL FOnPrivateMsg(PCS_MSGPRIVATE pMsg) { return TRUE; }
    virtual BOOL FOnInvite(PCS_MSGINVITE pcsInvite) { return TRUE; }
    // virtual BOOL FOnPropString(CSPROP_TYPE csType, char* sz) {
return TRUE; }
```

```

//      virtual BOOL      FOnPropBuffer(CSPROP_TYPE csType, BYTE* pbBuffer,
DWORD dwcb) { return TRUE; }
      virtual BOOL      FParseQueryData(PCS_PROPERTY pcsProp);

      // and if we got something we don't really handle..
      virtual BOOL      FUnknownMessage(PCS_MSGBASE pMsg)      { return TRUE; }

protected:
      static UINT      SocketThreadProc(LPVOID pvData);
      void      CleanUp();

      void      SetSocket(PICS pics);
      BOOL      FStartMessageThread();
      BOOL      FWaitForMessage();
      void      WaitForMsgThread();

      PICS      m_pics;          // typedef IChatSocket
      *PICS;
      PICS_FACTORY      m_pFactory; // typedef IChatSocketFactory
      *PICS_FACTORY;
      CWinThread*      m_pThread; // Message Thread.
      HANDLE      m_hThread; // Save the handle of the Thread

for ::WaitSingleObject
      BOOL      m_bCanceled;
      int      m_nPICSRef; // for Reference counting.

      BOOL      m_bQueryOK;          // Limit query if we didn't get the
answer yet...
      int      m_LastQType;          // What was the last query
type?
      int      m_iItem;          // current item
      HWND      m_hNotify; // window's handle to notify to.
};

#endif // _BASESOCK_H

```

```
// MainFrm.cpp : implementation of the CMainFrame class
//
```

```
#include "stdafx.h"
#include "MonTool.h"
#include "MainFrm.h"
#include "MonToolDoc.h"
#include "MonListView.h"
#include "MonToolView.h"
```

```
#include "pxmessages.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMainFrame
```

```
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
```

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_WM_TIMER()
    //}}AFX_MSG_MAP
    // User-Defined Messages
    //ON_MESSAGE(CSMMSG_CMD_ADDCHANNEL, OnCsAddChannel)
    //ON_MESSAGE(CSMMSG_CMD_PRIVATEMSG, OnCsPrivateMsg)
    ON_MESSAGE(CSMMSG_CMD_QUERYDATA, OnCsQueryData)
    ON_MESSAGE(CMD_QUERY_MEMBERS, OnCsQueryMembers)
    ON_MESSAGE(CMD_QUERY_MEMBERS_END, OnCsQueryMembersEnd)
    ON_MESSAGE(CMD_QUERY_CHANNELS, OnCsQueryChannels)
    ON_MESSAGE(CMD_QUERY_CHANNELS_END, OnCsQueryChannelsEnd)
    //ON_MESSAGE(CSMMSG_CMD_INVITE, OnCsInvite)
    //ON_MESSAGE(CSMMSG_CMD_GOTMEMLIST, OnCsGotMemList)
    //ON_MESSAGE(CSMMSG_CMD_ADDMEMBER, OnCsAddMember)
    //ON_MESSAGE(CSMMSG_CMD_DELMEMBER, OnCsDelMember)
    //ON_MESSAGE(CSMMSG_CMD_DELCHANNEL, OnCsDelChannel)
    //ON_MESSAGE(CSMMSG_CMD_MODEMEMBER, OnCsModeMember)
    //ON_MESSAGE(CSMMSG_CMD_MODECHANNEL, OnCsModeChannel)
    //ON_MESSAGE(CSMMSG_CMD_TEXT_A, OnCsTextA)
    //ON_MESSAGE(CSMMSG_CMD_DATA, OnCsData)
    //ON_MESSAGE(CSMMSG_CMD_WHISPERTEXT_A, OnCsWhisperText)
    //ON_MESSAGE(CSMMSG_CMD_WHISPERDATA, OnCsWhisperData)
    //ON_MESSAGE(CSMMSG_CMD_NEWTOPIC, OnCsNewTopic)
    //ON_MESSAGE(CSMMSG_CMD_NEWNICK, OnCsNewNick)
END_MESSAGE_MAP()
```

```
static UINT indicators[] =
{
    ID_SEPARATOR,          // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
```

};

```

////////////////////////////////////
// CMainFrame construction/destruction

```

CMainFrame::CMainFrame()

```

{
    // TODO: add member initialization code here
}

```

CMainFrame::~~CMainFrame()

```

{
}

```

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

```

{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;        // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;        // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizable
toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    // be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

```

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

```

{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
    cs.style &= -FWS_ADDTOTITLE;
    return CFrameWnd::PreCreateWindow(cs);
}

```

```

////////////////////////////////////
// CMainFrame diagnostics

```

```

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

```

```

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

```

```

#endif // _DEBUG

```

```

////////////////////////////////////
// CMainFrame message handlers

```

```

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpCS, CCreateContext*
pContext)
{
    // CG: The following block was added by the Split Bars component.
    /*
    {
        if (!m_wndSplitter.Create(this,
                                1, 2,          // TODO: adjust the
number of rows, columns
                                CSize(10, 10), // TODO: adjust the
minimum pane size
                                pContext))
        {
            TRACE0("Failed to create split bar ");
            return FALSE;    // failed to create
        }

        return TRUE;
    }
    */

    if (!m_wndSplitter.CreateStatic(this, 1, 2) ||
        !m_wndSplitter.CreateView(0, 1, RUNTIME_CLASS(CMonListView),
CSize(0, 0), pContext) ||
        !m_wndSplitter.CreateView(0, 0, RUNTIME_CLASS(CMonToolView),
CSize(300, 0), pContext))
        return FALSE;

    return TRUE;
}

```

```

////////////////////////////////////
// ChatSock Message Handler
LRESULT CMainFrame::OnCsQueryData(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMainFrame::OnCsQueryData\n");
}

```

```

    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsQueryData(wParam, lParam);
}

LRESULT CMainFrame::OnCsQueryMembers(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMainFrame::OnCsQueryData\n");

    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsQueryMembers(wParam, lParam);
}

LRESULT CMainFrame::OnCsQueryMembersEnd(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMainFrame::OnCsQueryData\n");

    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsQueryMembersEnd(wParam, lParam);
}

LRESULT CMainFrame::OnCsQueryChannels(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMainFrame::OnCsQueryData\n");

    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsQueryChannels(wParam, lParam);
}

LRESULT CMainFrame::OnCsQueryChannelsEnd(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMainFrame::OnCsQueryData\n");

    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsQueryChannelsEnd(wParam, lParam);
}
/*
LRESULT CMainFrame::OnCsAddChannel(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsAddChannel(wParam, lParam);
}

LRESULT CMainFrame::OnCsPrivateMsg(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsPrivateMsg(wParam, lParam);
}

LRESULT CMainFrame::OnCsInvite(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsInvite(wParam, lParam);
}

LRESULT CMainFrame::OnCsGotMemList(WPARAM wParam, LPARAM lParam)

```



```

{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsGotMemList(wParam, lParam);
}

LRESULT CMainFrame::OnCsAddMember(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsAddMember(wParam, lParam);
}

LRESULT CMainFrame::OnCsDelMember(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsDelMember(wParam, lParam);
}

LRESULT CMainFrame::OnCsDelChannel(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsDelChannel(wParam, lParam);
}

LRESULT CMainFrame::OnCsModeMember(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsModeMember(wParam, lParam);
}

LRESULT CMainFrame::OnCsModeChannel(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsModeChannel(wParam, lParam);
}

LRESULT CMainFrame::OnCsTextA(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsTextA(wParam, lParam);
}

LRESULT CMainFrame::OnCsData(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsData(wParam, lParam);
}

LRESULT CMainFrame::OnCsWhisperText(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsWhisperText(wParam, lParam);
}

LRESULT CMainFrame::OnCsWhisperData(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsWhisperData(wParam, lParam);
}

```

```
}

LRESULT CMainFrame::OnCsNewTopic(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsNewTopic(wParam, lParam);
}

LRESULT CMainFrame::OnCsNewNick(WPARAM wParam, LPARAM lParam)
{
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    return pDoc->OnCsNewNick(wParam, lParam);
}
*/

void CMainFrame::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    CMonToolDoc* pDoc = (CMonToolDoc*)GetActiveDocument();
    pDoc->Start();
    CFrameWnd::OnTimer(nIDEvent);
}
```

```

// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__565A73AB_A919_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_MAINFRM_H__565A73AB_A919_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "resource.h"

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Generated message map functions
protected:
    CSplitterWnd m_wndSplitter;
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext*
pContext);
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnTimer(UINT nIDEvent);
    //}}AFX_MSG
    // ChatSock
    //afx_msg LRESULT OnCsAddChannel(WPARAM, LPARAM);
    //afx_msg LRESULT OnCsPrivateMsg(WPARAM, LPARAM);
    afx_msg LRESULT OnCsQueryData(WPARAM, LPARAM);
    afx_msg LRESULT OnCsQueryMembers(WPARAM, LPARAM);

```

```

afx_msg LRESULT OnCsQueryMembersEnd(WPARAM, LPARAM);
afx_msg LRESULT OnCsQueryChannels(WPARAM, LPARAM);
afx_msg LRESULT OnCsQueryChannelsEnd(WPARAM, LPARAM);
//afx_msg LRESULT OnCsInvite(WPARAM, LPARAM);
//afx_msg LRESULT OnCsGotMemList(WPARAM, LPARAM);
//afx_msg LRESULT OnCsAddMember(WPARAM, LPARAM);
//afx_msg LRESULT OnCsDelMember(WPARAM, LPARAM);
//afx_msg LRESULT OnCsDelChannel(WPARAM, LPARAM);
//afx_msg LRESULT OnCsModeMember(WPARAM, LPARAM);
//afx_msg LRESULT OnCsModeChannel(WPARAM, LPARAM);
//afx_msg LRESULT OnCsTextA(WPARAM, LPARAM);
//afx_msg LRESULT OnCsData(WPARAM, LPARAM);
//afx_msg LRESULT OnCsWhisperText(WPARAM, LPARAM);
//afx_msg LRESULT OnCsWhisperData(WPARAM, LPARAM);
//afx_msg LRESULT OnCsNewTopic(WPARAM, LPARAM);
//afx_msg LRESULT OnCsNewNick(WPARAM, LPARAM);

DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_MAINFRM_H__565A73AB_A919_11D1_9169_0000F0610C92__INCLUDED_)

```

MonTool\MonListView

```
// MonListView.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "MonTool.h"
#include "MonToolDoc.h"
#include "MonListView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMonListView
```

```
IMPLEMENT_DYNCREATE(CMonListView, CListView)
```

```
CMonListView::CMonListView()
{
}
```

```
CMonListView::~CMonListView()
{
}
```

```
BEGIN_MESSAGE_MAP(CMonListView, CListView)
    //{AFX_MSG_MAP(CMonListView)
    // NOTE - the ClassWizard will add and remove mapping macros
    here.
    //}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CMonListView drawing
```

```
void CMonListView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
}
```

```
////////////////////////////////////
// CMonListView diagnostics
```

```
#ifdef _DEBUG
void CMonListView::AssertValid() const
{
    CListView::AssertValid();
}
```

```
void CMonListView::Dump(CDumpContext& dc) const
{
    CListView::Dump(dc);
}
```



```
// CMonListView message handlers
```

[illegible]

```

#ifdef AFX_MONLISTVIEW_H__565A73B7_A919_11D1_9169_0000F0610C92__INCLUDED_
#define AFX_MONLISTVIEW_H__565A73B7_A919_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MonListView.h : header file
//

////////////////////////////////////
// CMonListView view

class CMonListView : public CListView
{
protected:
    CMonListView();          // protected constructor used by dynamic
    creation
    DECLARE_DYNCREATE(CMonListView)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMonListView)
protected:
    virtual void OnDraw(CDC* pDC);    // overridden to draw this view
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CMonListView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{{AFX_MSG(CMonListView)
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

```

MonTool\MonListView

#endif //

!defined(AFX_MONLISTVIEW_H__565A73B7_A919_11D1_9169_0000F0610C92__INCLUDED_)

MonTool\MonTool.cpp

```
// MonTool.cpp : Defines the class behaviors for the application.
//
```

```
#include "stdafx.h"
#include "MonTool.h"
```

```
#include "MainFrm.h"
#include "MonToolDoc.h"
#include "MonToolView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMonToolApp
```

```
BEGIN_MESSAGE_MAP(CMonToolApp, CWinApp)
    //{AFX_MSG_MAP(CMonToolApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros
here.
        // DO NOT EDIT what you see in these blocks of generated code!
    }AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CMonToolApp construction
```

```
CMonToolApp::CMonToolApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```

```
////////////////////////////////////
// The one and only CMonToolApp object
```

```
CMonToolApp theApp;
```

```
////////////////////////////////////
// CMonToolApp initialization
```

```
BOOL CMonToolApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
```

```

        // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC
statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings(); // Load standard INI file options (including
MRU)

    // Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CMonToolDoc),
        RUNTIME_CLASS(CMainFrame),           // main SDI frame window
        RUNTIME_CLASS(CMonToolView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

    // Dialog Data
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    }}AFX_DATA

```

```

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
    // No message handlers
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CMonToolApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////
// CMonToolApp commands

```

MonTool\MonTool.h

```
// MonTool.h : main header file for the MONTOTOL application
//

#if !defined(AFX_MONTOTOL_H__82387668_B3A1_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_MONTOTOL_H__82387668_B3A1_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CMonToolApp:
// See MonTool.cpp for the implementation of this class
//

class CMonToolApp : public CWinApp
{
public:
    CMonToolApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMonToolApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CMonToolApp)
    afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions
    here.
    //      DO NOT EDIT what you see in these blocks of generated code
    !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#endif(AFX_MONTOTOL_H__82387668_B3A1_11D1_9169_0000F0610C92__INCLUDED_)
```

MonTool\MonToolDoc.

```
// MonToolDoc.cpp : implementation of the CMonToolDoc class
//
```

```
#include "stdafx.h"
#include "MonToolDoc.h"
#include "MonToolView.h"
#include "MonListView.h"
```

```
#include "resource.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMonToolDoc
```

```
IMPLEMENT_DYNCREATE(CMonToolDoc, CDocument)
```

```
BEGIN_MESSAGE_MAP(CMonToolDoc, CDocument)
    //{{AFX_MSG_MAP(CMonToolDoc)
    ON_COMMAND(ID_CONNECT, OnConnect)
    ON_UPDATE_COMMAND_UI(ID_CONNECT, OnUpdateConnect)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CMonToolDoc construction/destruction
```

```
CMonToolDoc::CMonToolDoc()
{
    // TODO: add one-time construction code here
    m_nMembers = 0;
    m_nChannels = 0;
    m_bStarted = FALSE;
    m_MonToolSet.Open();
}
```

```
CMonToolDoc::~CMonToolDoc()
{
    m_MonToolSet.Close();
}
```

```
BOOL CMonToolDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}
```

```

/////////////////////////////////////////////////////////////////
// CMonToolDoc serialization

void CMonToolDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////
// CMonToolDoc diagnostics

#ifdef _DEBUG
void CMonToolDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CMonToolDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

/////////////////////////////////////////////////////////////////
// CMonToolDoc commands
/////////////////////////////////////////////////////////////////

// return a pointer to the off-screen buffered view
CMonListView* CMonToolDoc::GetMonListView() const
{
    POSITION pos = GetFirstViewPosition();
    ASSERT(pos);
    CMonListView* pView = (CMonListView*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CMonListView)));
    return pView;
}

// return a pointer to the off-screen buffered view
CMonToolView* CMonToolDoc::GetMonToolView() const
{
    POSITION pos = GetFirstViewPosition();
    ASSERT(pos);
    CMonListView* pView = (CMonListView*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CMonListView)));
    CMonToolView* pToolView = (CMonToolView*)GetNextView(pos);
    ASSERT(pToolView);
    ASSERT(pToolView->IsKindOf(RUNTIME_CLASS(CMonToolView)));
}

```

```

        return pToolView;
    }

// ChatSock Message Handler: coming from CMainFrame
//
LRESULT CMonToolDoc::OnCsQueryData(WPARAM wParam, LPARAM lParam)
{
    int temp = (int)wParam;
    TRACE("temp = %d", temp);
    return 0;
}

LRESULT CMonToolDoc::OnCsQueryMembers(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMonToolDoc::OnCsQueryMembers\n");

    PICS_PROPERTY picsProp = (PICS_PROPERTY)lParam;
    ASSERT(picsProp);

    // Add the channel name..
    CS_PROPDATA cspd;
    if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_MEMBER_NAME)))
        return FALSE;
    CHAR* szName = (CHAR*)cspd.pbData;
    CString name(szName);
    GetMonToolView()->AddUsers(name);

    return 0;
}

LRESULT CMonToolDoc::OnCsQueryMembersEnd(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMonToolDoc::OnCsQueryMembersEnd\n");

    m_nMembers = (UINT)wParam;
    CString nTot;
    nTot.Format("Total Users : %d", m_nMembers);
    TRACE("Total Users : %d\n", m_nMembers);

    GetMonToolView()->AddUsers(nTot);
    GetMonToolView()->ExpandRoot();
    GetMonToolView()->ExpandUsers();

    OnQuerychannels();
    return 0;
}

LRESULT CMonToolDoc::OnCsQueryChannels(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMonToolDoc::OnCsQueryChannels\n");

    PICS_PROPERTY picsProp = (PICS_PROPERTY)lParam;
    ASSERT(picsProp);

    // Add the channel name..
    CS_PROPDATA cspd;
    if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_CHANNEL_NAME)))

```

```

        return FALSE;
    CHAR* szName = (CHAR*)cspd.pbData;

    if (FAILED(picsProp->HrGetProperty(&cspd, CSINDEX_PROP_CHANNEL_CUSER)))
        return FALSE;
    DWORD cUser = 1;
    cUser = *((DWORD *) cspd.pbData); // save it

    CString name(szName);
    CString num;
    num.Format(" (%d)", cUser);
    CString full = name + num;

    //GetMonToolView()->AddChannels(szName);
    GetMonToolView()->AddChannels(full);

    return 0;
}

LRESULT CMonToolDoc::OnCsQueryChannelsEnd(WPARAM wParam, LPARAM lParam)
{
    TRACE("CMonToolDoc::OnCsQueryChannelsEnd\n");

    m_nChannels = (UINT)wParam;
    CString nTot;
    nTot.Format("Total Channels : %d", m_nChannels);
    GetMonToolView()->AddChannels(nTot);
    nTot.Format("Total Users: (%d), Channels: (%d)", m_nMembers,
m_nChannels);
    GetMonToolView()->AddToRoot(nTot);

    GetMonToolView()->ExpandRoot();
    GetMonToolView()->ExpandChannels();

    WriteToTitle();

    // Write to DB.
    m_MonToolSet.AddNew();

    m_MonToolSet.m_Members1 = m_nMembers;
    m_MonToolSet.m_Channels = m_nChannels;
    m_MonToolSet.m_Time = COleDateTime::GetCurrentTime();

    m_MonToolSet.Update();

    return 0;
}

/*
// Since Document cannot receive WM_COMMAND message
// Frame window will call this function for us...
// typedef IChatSocketFactory* PICS_FACTORY;
// typedef IChatSocket* PICS;
// typedef ICSChannel* PICS_CHANNEL;
// typedef ICSMember* PICS_MEMBER;
// typedef ICSQuery* PICS_QUERY;

```



```

// typedef ICSProperty*          PICS_PROPERTY;
// typedef ICSPrivateMsg*        PICS_PRIVMSG;
// typedef ICSInvitation*        PICS_INVITATION;
//
// Typical sequence of commands from ChatSock
// 1. CSMSG_TYPE_ADDCHANNEL
// 2. CSMSG_TYPE_ADDMEMBER ...
// 3. CSMSG_TYPE_PRIVATEMSG
// 4. CSMSG_TYPE_GOTMEMLIST
// 5. CSMSG_TYPE_TEXT_A ...
// 6. CSMSG_TYPE_DELMEMBER ...
// 7. CSMSG_TYPE_DELCHANNEL

// Create my actor on ADDCHANNEL
LRESULT CMonToolDoc::OnCsAddChannel(WPARAM wParam, LPARAM lParam)
{
    PICS_CHANNEL picsChannel = (PICS_CHANNEL)lParam;          // ICSChannel*
    SetChannel(picsChannel);          // Don't Release pC here.

    ClearHistory();

    CString strTopic(m_pChannel->SzTopic());
    int nID = GetEncodedStageID(strTopic);
    LoadStage(nID);

    picsChannel->Release();

    CString strMsg;
#ifdef _ENGLISH
    strMsg.Format("[%s] joined this channel.", m_pChannel->SzName());
#else
    strMsg.Format("[%s] 'eÈ-!æ¿; µé¼í ¿Ô¼Ä'Í'Û.", m_pChannel->SzName());
#endif
    // GetView()->ChatVoice(strMsg);
    DisplayText(strMsg);

    return 0;
}

// When I enter a channel, I receive PX member info via Private message
LRESULT CMonToolDoc::OnCsPrivateMsg(WPARAM wParam, LPARAM lParam)
{
    /*
    PICS_PRIVMSG picsPrivMsg = (PICS_PRIVMSG)lParam;          // ICSPrivateMsg*
    ASSERT(picsPrivMsg);
    CS_PRIVMSG pm;
    if (FAILED(picsPrivMsg->HrGetMsg(&pm)))
    {
        TRACE("picsPrivMsg->HrGetMsg() failed!\n");
        picsPrivMsg->Release();
        return FALSE;
    }
    CString strData;
    if (pm.fText)
    {
        if (pm.fAnsi)

```

```

    {
        TRACE0("CSMSG_CMD_PRIVATEMSG - not ANSI\n");
        picsPrivMsg->Release();
#ifdef _ENGLISH
        strData.Format("Private Message:\'%s'", (CHAR*)pm.pbData);
#else
        strData.Format("'°ñ'D '»¿e:\'%s'", (CHAR*)pm.pbData);
#endif
        DisplayText(strData);
        return FALSE;
    }
else // binary
{
    TRACE("Private MSG: binary\n");
    parser.CopyBuffer((CHAR*)pm.pbData);
    char c;
    parser.GetValueRightToken(c, TOKEN);
    if (c == 'X')
    {
        char* szNickFrom;
        BOOL bAnsi;
        picsPrivMsg->HrGetMsgSender((BYTE**) &szNickFrom, &bAnsi);
        PICS_MEMBER pM=NULL;
        if (szNickFrom && m_pChannel)
        {
            PICS_CHANNEL picsChannel = m_pChannel->PChannel();
            ASSERT(picsChannel);
            picsChannel->HrGetMemberFromNameA(szNickFrom, &pM);
            picsChannel->Release();

            CActor* pA = GetActor(pM); // Find corresponding
            Actor object
            int nCmd;
            strData = parser.GetValueRightToken(nCmd, TOKEN);
            if (nCmd == CMD_MEMBER_INFO)
            {
                TRACE("\tCMD_MEMBER_INFO: %lx\n", pM);
                if (!pA) // Not Found: New User
                {
                    CMemberInfo mi;
                    mi.SetMember(pM);
                    if (mi.SetMemberInfo(strData) &&
m_pStage)
                        {
pA = m_pStage->CreateActor(mi,
FALSE); // m_pChannel->FIsMemberMe(pM));
                        }
                    }
                }
            if (pM)
                pM->Release();
        }
    }
    picsPrivMsg->Release();
}

```

```

        return 0;
    }

LRESULT CMonToolDoc::OnCsInvite(WPARAM wParam, LPARAM lParam)
{
    /*
    PICS_INVITATION picsInvite = (PICS_INVITATION)lParam;
    ASSERT(picsInvite);
    BYTE* pbName;
    BYTE* pbSender;
    BOOL bAnsi;
    CHAR* szChannel;
    CHAR* szFrom;

    if (FAILED(picsInvite->HrGetChannelName(&pbName, &bAnsi)))
    {
        picsInvite->Release();
        return FALSE;
    }

    if (bAnsi)
    {
        szChannel = (CHAR*)pbName;
    }

    if (FAILED(picsInvite->HrGetSender(&pbSender, &bAnsi)))
    {
        picsInvite->Release();
        return FALSE;
    }

    if (bAnsi)
    {
        szFrom = (CHAR*)pbSender;
    }

    CString msg;
#ifdef _ENGLISH
    msg.Format("<%s> invited you to [%s].\n"
               "Do you want to join?", szFrom, szChannel);
#else
    msg.Format("[%s] 'eÈ-¹æ¿;¼- <%s> invited.\n"
               "ÄÊ'e¿; ÄÄÇI¼Ä°Ü¼Ä'İ±i?", szChannel, szFrom);
#endif
    if (AfxMessageBox(msg, MB_YESNO) == IDYES)
    {
        ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_COMMAND,
        (WPARAM) IDC_PANEL_BUTTON, (LPARAM) 0);
    }
    picsInvite->Release();

    return 0;
}

```

```

////////////////////////////////////

```

```

// Channel thread

LRESULT CMonToolDoc::OnCsGotMemList(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

// We only respond to the ADDMEMBER message after GOTMEMLIST
// 1. On ADDMEMBER myself
//      SendData => Inform myself to this channel I just joined
// 2. On ADDMEMBER new member in the course of chat
//      SendPrivData => Inform myself only to this new member
LRESULT CMonToolDoc::OnCsAddMember(WPARAM wParam, LPARAM lParam)
{
    /*
    ASSERT(m_pChannel);
    PICS_MEMBER pM = (PICS_MEMBER)lParam;
    ASSERT(pM);

    CHAR* szName = m_pChannel->SzMemName(pM);
    //AddMember(picsMember);
    CString strData;
    CString str;
#ifdef _ENGLISH
    str.Format("[%s] entered.", szName);
#else
    str.Format("[%s]´Ô µiÀÄ.", szName);
#endif
    DisplayText(str);
    // Member info
    if (m_pChannel->FGotMemList()) // After
    {
        if (m_pChannel->FIsMemberMe(pM)) // Broadcast my info
        {
            // Load my actor
            CMemberInfo mi;
            mi.LoadMyInfo();
            CRect rc;
            m_pStage->GetBase(rc);
            rc.InflateRect(-20, -20, -20, -20);
            srand((unsigned)time(NULL));
            mi.SetBase(CPoint(rc.left + rand() % rc.Width(), rc.top +
rand() % rc.Height()));

            PICS_CHANNEL pC = m_pChannel->PChannel();
            ASSERT(pC);
            PICS_MEMBER pM;
            pC->HrGetMe(&pM);
            ASSERT(pM);
            pC->Release();
            mi.SetMember(pM); // AddRef()
            pM->Release();
            mi.SetVersion(GetClientVersion()); // Myself

            CActor* pA = m_pStage->CreateActor(mi, TRUE); // myself
            CActor* pA = GetActor(pM);
            if (pA)

```

```

        {
            strData.Format("X`%d`", CMD_MEMBER_INFO);
            pA->m_MI.GetMemberInfo(str);
            strData += str;
            TRACE("OnCsAddMember: [SendData] %s\n", strData);
            SendData(strData);
        }
        else
        {
            TRACE0("CMonToolDoc::OnCsAddMember - Creation
Failure!\n");
        }
    }
    else // Inform myself to the new member
    {
        CActor* pA = GetThisActor();
        if (pA)
        {
            strData.Format("X`%d`", CMD_MEMBER_INFO);
            CMemberInfo mi;
            pA->GetMemberInfo(mi); // Get member info from this
actor
string
            mi.GetMemberInfo(str); // express member info in

            strData += str;
            DWORD dwcb = strData.GetLength() + 1;
            TRACE("OnCsAddMember: [PrivData] %s\n", strData);
            m_pSocket->FSendPrivData(szName,
(BYTE*)strData.GetBuffer(dwcb-1), dwcb);
        }
        else
        {
            TRACE0("CMonToolDoc::OnCsAddMember - Oops!, I'm not
created yet?\n");
        }
    }
    pM->Release();

    return 0;
}

LRESULT CMonToolDoc::OnCsDelMember(WPARAM wParam, LPARAM lParam)
{
    /*
    PCS_MSGMEMBER pMsg = (PCS_MSGMEMBER)lParam;
    PICS_MEMBER pM = pMsg->picsMember;
    ASSERT(pM);
    pM->AddRef();

    if (m_pChannel)
    {
        CHAR* szName = m_pChannel->SzMemName(pM);
        //DelMember(picsMember);
        CString str;
#ifdef _ENGLISH
        str.Format("[%s] out", szName);

```

```

#else
    str.Format("[%s] 'Ö ÅðÄä", szName);
#endif

    DisplayText(str);

    // Delete the member
    CActor* pA = GetActor(pM);

    if (pMsg->picsMemSrc)
    {
        PICS_MEMBER pMemKick = pMsg->picsMemSrc;
        // Someone was kicked..
        // If the member being kicked was US.. notify the user
        if (m_pChannel->FIsMemberMe(pM)) // This doesn't work
        since the member is already kicked out.
        {
            CString strKickReason;
            CHAR* szKicker = m_pChannel->SzMemName(pMemKick);
            // Also save the reason, if any.. and only if its ANSI
            if (pMsg->pvReason && pMsg->fAnsi)
            {
                if (szKicker && pMsg->pvReason)
                {
#ifdef _ENGLISH
                    strKickReason.Format("(Kick Off) %s kicked off
                    %s!\r\n(Reason) %s",
                #else
                    strKickReason.Format("(°-Åð) %s'ÔÀÌ %s'ÔÀ»
                    Å³ ÅÄ¼!\r\n(»çÄ~) %s",
                #endif
                    szKicker, szName, (CHAR*)pMsg->pvReason);

                    DisplayText(strKickReason);
                    if (pA == GetThisActor())
                    {
                        AfxMessageBox(strKickReason);
                    }
                }
            }

            if (pA)
            {
                delete m_pStage->RemoveActor(pA);
                CRect rc;
                m_pStage->GetRect(rc);
                GetView()->AddDirtyRegion(&rc);
                GetView()->RenderAndDrawDirtyList(); // Erase
            }
        }
        pM->Release();

        return 0;
    }
}

LRESULT CMonToolDoc::OnCsDelChannel(WPARAM wParam, LPARAM lParam)
{
    /*

```

```

        SetTitle(NULL);
        PICS_CHANNEL pC = (PICS_CHANNEL)lParam;
        ASSERT(pC);
        TRACE("OnCsDelChannel\n");
        pC->Release();

        return 0;
    }

LRESULT CMonToolDoc::OnCsModeMember(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

LRESULT CMonToolDoc::OnCsModeChannel(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

// On TEXT_A - Just a text conversation
// 1. Find the actor object corresponding to the MIC member
// 2. Show the text through this actor
LRESULT CMonToolDoc::OnCsTextA(WPARAM wParam, LPARAM lParam)
{
    /*

        PCS_MSG pMsg = (PCS_MSG)lParam;
        // Obtain the sender's name
        PICS_MEMBER pM = pMsg->picsFrom;
        ASSERT(m_pChannel);
        // if member is NULL, the message is from the channel
        CHAR* szName = (pM) ? m_pChannel->SzMemName(pM) : m_pChannel->SzName();
        ASSERT(szName);
        // Is this user a Host?
        CString strText;
        if (pM->HrIsMemberHost() == NOERROR)
            strText = _T("");
        CString strVoice((LPCSTR)pMsg->pbData);

        // Registration Serivece by Jun
        if(strVoice == "XREGISTEREDX")
            AfxMessageBox("You are registered member.");

        //

// if (StripComicHeader((LPCSTR)pMsg->pbData, strVoice) > 0)
// {
    strText += szName;
    strText += _T(":");
    strText += strVoice;
    DisplayText(strText);

    if (pM->HrIsMemberIgnored() == NOERROR)
        return 0;

    CActor* pA = GetActor(pM);

```

```

    if (pA)
    {
        if (strText.GetLength() > MAX_SEND_CHAR)
        {
            strText.GetBufferSetLength(MAX_SEND_CHAR);
            strText.ReleaseBuffer(MAX_SEND_CHAR);
        }
        pA->m_MI.SetBubbleKind(0);    // Yello
        pA->Chat(strText);            // Later, consider DBCS...
        if (pA->IsVoice())
            GetView()->ChatVoice(strVoice, pA->GetGender());
    }
    //
    return 0;
}

// 1. Actors' command data
LRESULT CMonToolDoc::OnCsData(WPARAM wParam, LPARAM lParam)
{
    /*
    PCS_MSG pMsg = (PCS_MSG)lParam;
    // Obtain the sender's name
    PICS_MEMBER pM = pMsg->picsFrom;

    if (pM->HrIsMemberIgnored() == NOERROR)
        return 0;
    TRACE0("Data: binary\n");
    int len = (int)pMsg->dwcbData;
    char* szBuf = new char[len+2];    // +1 for safety
    ::CopyMemory(szBuf, (LPBYTE)pMsg->pbData, len);
    szBuf[len] = NULL;
    parser.CopyBuffer(szBuf);
    char c;
    parser.GetValueRightToken(c, TOKEN);
    if (c == 'X')
    {
        CActor* pA = GetActor(pM);    // Find corresponding Actor object
        int nCmd;
        CString strData(parser.GetValueRightToken(nCmd, TOKEN));
        if (nCmd == CMD_MEMBER_INFO)
        {
            TRACE("\tCMD_MEMBER_INFO: %lx\n", pM);
            if (!pA)    // Not Found: New Member
            {
                CMemberInfo mi;
                mi.SetMember(pM);
                if (mi.SetMemberInfo(strData) && m_pStage)
                {
                    CActor* pA = m_pStage->CreateActor(mi,
m_pChannel->FIsMemberMe(pM));
                }
            }
        }
        else if (nCmd > CMD_MOVE)    // Move or Action
        {
            // CMD_ACTION means
just repositioning

```



```

        TRACE("\tCMD_MOVE_ or ACTION(%d): %lx => ", nCmd, pM);
        if (pA)
        {
            int          nHDir;
            CPoint        ptBase;
            if (parser.GetValueRightToken(ptBase, TOKEN))
            {
                parser.GetValueRightToken(nHDir, TOKEN);
                pA->SetState(ptBase, nHDir); // Adjust Current
State
            }
            pA->Action(nCmd);
        }
    }
    delete [] szBuf;

    return 0;
}

LRESULT CMonToolDoc::OnCsWhisperText(WPARAM wParam, LPARAM lParam)
{
    /*
    PCS_MSGWHISPER pMsgW = (PCS_MSGWHISPER)lParam;
    ASSERT(pMsgW);

    PCS_MSG pMsg = (PCS_MSG)pMsgW->pcsMsg;
    ASSERT(pMsg);

    PICS_MEMBER pM = pMsg->picsFrom;
    ASSERT(m_pChannel);
    // if member is NULL, the message is from the channel
    CHAR* szName = (pM) ? m_pChannel->SzMemName(pM) : m_pChannel->SzName();
    ASSERT(szName);
    // Is this user a Host?
    CString strText;
    if (pM->HrIsMemberHost() == NOERROR)
        strText = _T("");
    CString strVoice((LPCSTR)pMsg->pbData);
    strText += szName;
    strText += _T("=>");
    strText += strVoice;
    DisplayText(strText);

    if (pM->HrIsMemberIgnored() == NOERROR) // This member is ignored...
        return 0;

    CActor* pA = GetActor(pM);
    if (pA)
    {
        if (strText.GetLength() > MAX_SEND_CHAR)
        {
            strText.GetBufferSetLength(MAX_SEND_CHAR);
            strText.ReleaseBuffer(MAX_SEND_CHAR);
        }
        pA->m_MI.SetBubbleKind(1); // Green
        pA->Chat(strText, FALSE); // Later, consider DBCS...
    }
}

```

```

        if (pA->IsVoice())
            GetView()->ChatVoice(strVoice, pA->GetGender());
    }

    return 0;
}

LRESULT CMonToolDoc::OnCsWhisperData(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

LRESULT CMonToolDoc::OnCsNewTopic(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

LRESULT CMonToolDoc::OnCsNewNick(WPARAM wParam, LPARAM lParam)
{
    return 0;
}

*/

void CMonToolDoc::OnQueryAllUsers()
{
    // TODO: Add your command handler code here
    TRACE("CMonToolDoc::OnQueryAllUsers\n");

    m_nMembers = 0;
    GetMonToolView()->GetTreeCtrl().DeleteAllItems();
    GetMonToolView()->NewQueryStart();

    if(!m_Socket.FQueryListAllUsers())
    {
        AfxMessageBox("Fail to query ListAllUsers");
    }
}

BOOL CMonToolDoc::Connect()
{
    if(!m_Socket.FInit())
        return FALSE;

    EC_CONNINFO cInfo;
    cInfo.szServer = "88.1.26.2"; //noah server
    // cInfo.szServer = "70.2.173.70"; //monnanee
    cInfo.szNick = "UniChat2MonTool";
    cInfo.szNickBak = "UC2MonTool";
    cInfo.szUserName = "UniChat2 Monitoring Tool";
    cInfo.szPass = NULL;
    cInfo.fAuthenticate = FALSE;
    cInfo.dwTimeOut = 10000; // 10seconds.

    if(!m_Socket.FConnect(&cInfo, AfxGetMainWnd()->GetSafeHwnd()))
    {

```

```

        return FALSE;
    }

    return TRUE;
}

void CMonToolDoc::OnQuerychannels()
{
    // TODO: Add your command handler code here
    TRACE("CMonToolDoc::OnQuerychannels\n");

    m_nChannels = 0;
    //GetMonToolView()->GetTreeCtrl().DeleteAllItems();
    //GetMonToolView()->NewQueryStart();
    if(!m_Socket.FQueryListChannels())
    {
        TRACE("Failed to query ListChannels\n");
        AfxMessageBox("Failed to query ListChannels");
    }
}

void CMonToolDoc::OnConnect()
{
    // TODO: Add your command handler code here
    if(!Connect())
    {
        AfxMessageBox("Connection Failed!");
        return;
    }

    m_bStarted = TRUE;

    //AfxGetMainWnd()->SetTimer(1, 300000, NULL);
    AfxGetMainWnd()->SetTimer(1, 60000, NULL);
}

void CMonToolDoc::OnQueryall()
{
}

void CMonToolDoc::WriteToTitle()
{
    CString title;
    title.Format("UniChat Monitoring Tool : Total Channels : %d(s), Users : %d(s)", m_nChannels, m_nMembers);
    AfxGetMainWnd()->SetWindowText(title);
}

void CMonToolDoc::Start()
{
    TRACE("CMonToolDoc::Start\n");

    OnQueryAllUsers();
}

```

MonTool\MonToolDoc.c

```
void CMonToolDoc::OnUpdateConnect (CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(!m_bStarted);
}
```

```

// MonToolDoc.h : interface of the CMonToolDoc class
//
/////////////////////////////////////////////////////////////////

#ifdef
!defined(AFX_MONTOOLDOC_H__565A73AD_A919_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_MONTOOLDOC_H__565A73AD_A919_11D1_9169_0000F0610C92__INCLUDED_

#include "BaseSock.h"    // Added by ClassView
#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "MonToolSet.h"

class CMonToolView;
class CMonListView;

class CMonToolDoc : public CDocument
{
protected: // create from serialization only
    CMonToolDoc();
    DECLARE_DYNCREATE(CMonToolDoc)

// Attributes
public:
    CMonToolView*      GetMonToolView() const;
    CMonListView*      GetMonListView() const;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CMonToolDoc)
    public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMonToolDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

public:
    void Start();
    void WriteToTitle();
    // ChatSock messages
    //LRESULT OnCsAddChannel(WPARAM, LPARAM);

```

```

//LRESULT OnCsPrivateMsg(WPARAM, LPARAM);
LRESULT OnCsQueryData(WPARAM, LPARAM);
LRESULT OnCsQueryMembers(WPARAM, LPARAM);
LRESULT OnCsQueryMembersEnd(WPARAM, LPARAM);
LRESULT OnCsQueryChannels(WPARAM, LPARAM);
LRESULT OnCsQueryChannelsEnd(WPARAM, LPARAM);
//LRESULT OnCsInvite(WPARAM, LPARAM);
//LRESULT OnCsGotMemList(WPARAM, LPARAM);
//LRESULT OnCsAddMember(WPARAM, LPARAM);
//LRESULT OnCsDelMember(WPARAM, LPARAM);
//LRESULT OnCsDelChannel(WPARAM, LPARAM);
//LRESULT OnCsModeMember(WPARAM, LPARAM);
//LRESULT OnCsModeChannel(WPARAM, LPARAM);
//LRESULT OnCsTextA(WPARAM, LPARAM);
//LRESULT OnCsData(WPARAM, LPARAM);
//LRESULT OnCsWhisperText(WPARAM, LPARAM);
//LRESULT OnCsWhisperData(WPARAM, LPARAM);
//LRESULT OnCsNewTopic(WPARAM, LPARAM);
//LRESULT OnCsNewNick(WPARAM, LPARAM);

// Generated message map functions
protected:
    CMonToolSet m_MonToolSet;
    BOOL m_bStarted;
    UINT m_nChannels;
    UINT m_nMembers;
    BOOL Connect();
    CBaseSocket m_Socket;
    //{AFX_MSG(CMonToolDoc)
    afx_msg void OnQueryAllUsers();
    afx_msg void OnQuerychannels();
    afx_msg void OnConnect();
    afx_msg void OnQueryall();
    afx_msg void OnUpdateConnect(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_MONTOOLDOC_H__565A73AD_A919_11D1_9169_0000F0610C92__INCLUDED_)

```

```
// MonToolSet.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "MonToolSet.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMonToolSet
```

```
IMPLEMENT_DYNAMIC(CMonToolSet, CDaoRecordset)
```

```
CMonToolSet::CMonToolSet(CDaoDatabase* pdb)
: CDaoRecordset(pdb)
```

```
{
    //{AFX_FIELD_INIT(CMonToolSet)
    m_ID = 0;
    m_Channels = 0;
    m_Members1 = 0;
    m_Members2 = 0;
    m_Reserved1 = _T("");
    m_Reserved2 = _T("");
    m_nFields = 7;
    //}}AFX_FIELD_INIT
    m_nDefaultType = dbOpenTable;
}
```

```
CString CMonToolSet::GetDefaultDBName()
```

```
{
    return _T("MonTool.mdb");
}
```

```
CString CMonToolSet::GetDefaultSQL()
```

```
{
    return _T("[MonTool]");
}
```

```
void CMonToolSet::DoFieldExchange(CDaoFieldExchange* pFX)
```

```
{
    //{AFX_FIELD_MAP(CMonToolSet)
    pFX->SetFieldType(CDaoFieldExchange::outputColumn);
    DFX_Long(pFX, _T("[ID]"), m_ID);
    DFX_DateTime(pFX, _T("[Time]"), m_Time);
    DFX_Long(pFX, _T("[Channels]"), m_Channels);
    DFX_Long(pFX, _T("[Members1]"), m_Members1);
    DFX_Long(pFX, _T("[Members2]"), m_Members2);
    DFX_Text(pFX, _T("[Reserved1]"), m_Reserved1);
    DFX_Text(pFX, _T("[Reserved2]"), m_Reserved2);
    //}}AFX_FIELD_MAP
}
```

```

////////////////////////////////////
// CMonToolSet diagnostics

#ifdef _DEBUG
void CMonToolSet::AssertValid() const
{
    CDaoRecordset::AssertValid();
}

void CMonToolSet::Dump(CDumpContext& dc) const
{
    CDaoRecordset::Dump(dc);
}
#endif // _DEBUG

```



```

#if
!defined(AFX_MONTOOLSET_H__B1C91323_B1EF_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_MONTOOLSET_H__B1C91323_B1EF_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MonToolSet.h : header file
//

////////////////////////////////////
// CMonToolSet DAO recordset

class CMonToolSet : public CDaoRecordset
{
public:
    CMonToolSet(CDaoDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(CMonToolSet)

// Field/Param Data
    //{AFX_FIELD(CMonToolSet, CDaoRecordset)
    long m_ID;
    COleDateTime m_Time;
    long m_Channels;
    long m_Members1;
    long m_Members2;
    CString m_Reserved1;
    CString m_Reserved2;
    //}AFX_FIELD

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CMonToolSet)
    public:
        virtual CString GetDefaultDBName(); // Default database name
        virtual CString GetDefaultSQL(); // Default SQL for Recordset
        virtual void DoFieldExchange(CDaoFieldExchange* pFX); // RFX support
    //}AFX_VIRTUAL

// Implementation
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
};

//{AFX_INSERT_LOCATION}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_MONTOOLSET_H__B1C91323_B1EF_11D1_9169_0000F0610C92__INCLUDED_)

```

MonTool\MonToolView

```
// MonToolView.cpp : implementation of the CMonToolView class
//
```

```
#include "stdafx.h"
#include "MonTool.h"
#include "MonToolDoc.h"
#include "MonToolView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMonToolView
```

```
IMPLEMENT_DYNCREATE(CMonToolView, CTreeView)
```

```
BEGIN_MESSAGE_MAP(CMonToolView, CTreeView)
    //{AFX_MSG_MAP(CMonToolView)
    ON_WM_CREATE()
    //}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CTreeView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CTreeView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CTreeView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CMonToolView construction/destruction
```

```
CMonToolView::CMonToolView()
{
    // TODO: add construction code here
}
```

```
CMonToolView::~CMonToolView()
{
}
```

```
BOOL CMonToolView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    cs.style |= TVS_HASLINES | TVS_LINESATROOT | TVS_HASBUTTONS |
TVS_SHOWSELALWAYS;

    return CTreeView::PreCreateWindow(cs);
}
```

```
////////////////////////////////////
// CMonToolView drawing
```

```
void CMonToolView::OnDraw(CDC* pDC)
```

```

{
    CMonToolDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: add draw code for native data here

    GetTreeCtrl().Expand(m_hRoot, TVE_EXPAND);
    GetTreeCtrl().Expand(m_hUsers, TVE_EXPAND);
    GetTreeCtrl().Expand(m_hChannels, TVE_EXPAND);
}

void CMonToolView::OnInitialUpdate()
{
    CTreeView::OnInitialUpdate();

    // TODO: You may populate your TreeView with items by directly
    // accessing
    // its tree control through a call to GetTreeCtrl().
}

////////////////////////////////////
// CMonToolView printing
////////////////////////////////////

BOOL CMonToolView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CMonToolView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CMonToolView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

////////////////////////////////////
// CMonToolView diagnostics
////////////////////////////////////

#ifdef _DEBUG
void CMonToolView::AssertValid() const
{
    CTreeView::AssertValid();
}

void CMonToolView::Dump(CDumpContext& dc) const
{
    CTreeView::Dump(dc);
}

CMonToolDoc* CMonToolView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMonToolDoc)));
}

```

```

        return (CMonToolDoc*)m_pDocument;
    }
#endif // _DEBUG

////////////////////////////////////
// CMonToolView message handlers

int CMonToolView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CTreeView::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO: Add your specialized creation code here
    m_imgList.Create(IDB_BITMAP1, 16, 1, RGB(255, 0, 255));

    GetTreeCtrl().SetImageList(&m_imgList, TVSIL_NORMAL);

    InitTree();
    return 0;
}

int CMonToolView::InitTree()
{
    // Root Item first
    m_hRoot = GetTreeCtrl().InsertItem("UniChat! Incredible World!",
TVI_ROOT);
    m_hUsers = GetTreeCtrl().InsertItem("Users", m_hRoot);
    m_hChannels = GetTreeCtrl().InsertItem("Channels", m_hRoot);
    return 0;
}

void CMonToolView::AddUsers(CString name)
{
    GetTreeCtrl().InsertItem(name, 2, 2, m_hUsers);
}

void CMonToolView::AddChannels(CString channels)
{
    GetTreeCtrl().InsertItem(channels, 4, 4, m_hChannels);
}

void CMonToolView::ExpandUsers()
{
    GetTreeCtrl().Expand(m_hUsers, TVE_EXPAND);
}

void CMonToolView::ExpandChannels()
{
    GetTreeCtrl().Expand(m_hChannels, TVE_EXPAND);
}

void CMonToolView::ExpandRoot()
{
    GetTreeCtrl().Expand(m_hRoot, TVE_EXPAND);
}

void CMonToolView::NewQueryStart()

```

MonTool\MonToolView

```
{
    m_hRoot = GetTreeCtrl().InsertItem("UniChat! Incredible World!",0, 0);
    m_hUsers = GetTreeCtrl().InsertItem("Users",1, 1,m_hRoot);
    m_hChannels = GetTreeCtrl().InsertItem("Channels",3,3, m_hRoot);
}

void CMonToolView::AddToRoot(CString str)
{
    GetTreeCtrl().InsertItem(str,0, 0, m_hRoot);
}
```

MonTool\MonToolView

```
// MonToolView.h : interface of the CMonToolView class
//
///////////////////////////////////////////////////////////////////

#ifdef _AFX_
#ifndef AFX_MONTOOLVIEW_H__565A73AF_A919_11D1_9169_0000F0610C92__INCLUDED_
#define AFX_MONTOOLVIEW_H__565A73AF_A919_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMonToolDoc;

class CMonToolView : public CTreeView
{
protected: // create from serialization only
    CMonToolView();
    DECLARE_DYNCREATE(CMonToolView)

// Attributes
public:
    CMonToolDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMonToolView)
public:
    virtual void OnDraw(CDC* pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void OnInitialUpdate(); // called first time after construct
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
    void AddToRoot(CString str);
    void NewQueryStart();
    void ExpandRoot();
    void ExpandChannels();
    void ExpandUsers();
    void AddChannels(CString channels);
    void AddUsers(CString name);
    virtual ~CMonToolView();

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

```

```
// Generated message map functions
protected:
```

```
    HTREEITEM m_hChannels;
    HTREEITEM m_hUsers;
    HTREEITEM m_hRoot;
    int InitTree();
    CImageList m_imgList;
    //{AFX_MSG(CMonToolView)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

```
};
```

```
#ifndef _DEBUG // debug version in MonToolView.cpp
```

```
inline CMonToolDoc* CMonToolView::GetDocument()
```

```
{ return (CMonToolDoc*)m_pDocument; }
```

```
#endif
```

```
////////////////////////////////////
```

```
//{AFX_INSERT_LOCATION}}
```

```
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
```

```
#endif //
```

```
!defined(AFX_MONTOOLVIEW_H__565A73AF_A919_11D1_9169_0000F0610C92__INCLUDED_)
```

=====

MICROSOFT FOUNDATION CLASS LIBRARY : MonTool

=====

AppWizard has created this MonTool application for you. This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that make up your MonTool application.

MonTool.h

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CMonToolApp application class.

MonTool.cpp

This is the main application source file that contains the application class CMonToolApp.

MonTool.rc

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

res\MonTool.ico

This is an icon file, which is used as the application's icon. This icon is included by the main resource file MonTool.rc.

res\MonTool.rc2

This file contains resources that are not edited by Microsoft Developer Studio. You should place all resources not editable by the resource editor in this file.

MonTool.clw

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

////////////////////////////////////

For the main frame window:

MainFrm.h, MainFrm.cpp

These files contain the frame class CMainFrame, which is derived from CFrameWnd and controls all SDI frame features.

res\Toolbar.bmp

This bitmap file is used to create tiled images for the toolbar. The initial toolbar and status bar are constructed in the CMainFrame class. Edit this toolbar bitmap along with the array in MainFrm.cpp to add more toolbar buttons.

////////////////////////////////////////////////////////////////

AppWizard creates one document type and one view:

MonToolDoc.h, MonToolDoc.cpp - the document

These files contain your CMonToolDoc class. Edit these files to add your special document data and to implement file saving and loading (via CMonToolDoc::Serialize).

MonToolView.h, MonToolView.cpp - the view of the document

These files contain your CMonToolView class.
CMonToolView objects are used to view CMonToolDoc objects.

////////////////////////////////////////////////////////////////

Other standard files:

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header (PCH) file named MonTool.pch and a precompiled types file named StdAfx.obj.

Resource.h

This is the standard header file, which defines new resource IDs. Microsoft Developer Studio reads and updates this file.

////////////////////////////////////////////////////////////////

Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you should add to or customize.

If your application uses MFC in a shared DLL, and your application is in a language other than the operating system's current language, you will need to copy the corresponding localized resources MFC40XXX.DLL from the Microsoft Visual C++ CD-ROM onto the system or system32 directory, and rename it to be MFCLOC.DLL. ("XXX" stands for the language abbreviation. For example, MFC40DEU.DLL contains resources translated to German.) If you don't do this, some of the UI elements of your application will remain in the language of the operating system.

////////////////////////////////////////////////////////////////

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by MonTool.rc
//
#define IDD_ABOUTBOX 100
#define IDR_MAINFRAME 128
#define IDR_MONTOTOYPE 129
#define IDB_BITMAP1 130
#define IDB_BITMAP4 133
#define IDB_BITMAP2 143
#define ID_CONNECT 32774

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS 1
#define _APS_NEXT_RESOURCE_VALUE 144
#define _APS_NEXT_COMMAND_VALUE 32776
#define _APS_NEXT_CONTROL_VALUE 1000
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif
```

MonTool\StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//     MonTool.pch will be the pre-compiled header
//     stdafx.obj will contain the pre-compiled type information
```

```
#include "stdafx.h"
```

```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H__565A73A9_A919_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_STDAFX_H__565A73A9_A919_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>             // MFC core and standard components
#include <afxext.h>             // MFC extensions
#include <afxview.h>
#include <afxdisp.h>           // MFC OLE automation classes
#include <afxdao.h>
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__565A73A9_A919_11D1_9169_0000F0610C92__INCLUDED_)

```

```

//
=====
// File: P R O G R E S S . C P P
//
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
// PARTICULAR PURPOSE.
//
// Description:
//
// This sample demonstrates using a URL moniker to download information.
// The key routines include the implementation of IBindStatusCallback
// and the CDownload::DoDownload routine, which creates and binds to the
// URL moniker.
//
// Instructions:
//
// To use this sample:
// * build it using the NMAKE command. NMAKE will create PROGRESS.EXE.
// * run PROGRESS.EXE. specify the resource to download by passing an
//   URL on the command-line. use no command-line argument to default to
//   downloading "http://www.msn.com".
// * The program displays a dialog box containing information about the
//   download:
//   - a status message, describing the current status of the download
//   - a progress message, describing the amount of information that
//     has been downloaded.
//   - a text box, which displays chunks of the download information as
//     it arrives.
// * Press the "GO" button to begin the download.
//
// Sample update:-
// * New feature include Progress Bar to indicate progress of the download
// * Edit box replaces the old text box. Also the whole file can be
//   viewed.
//   If the file exceeds 32KB then only the first 32 KB from the last data
//   pull will be displayed. If the pull exceeds 32 KB, then only the
//   first
//   32 KB of the last Read will be displayed.
//
//
//
// File updated by Jason Strayer 27-Aug-1997
// File updated by Jobi George 19-June-1996
// File updated by Ramesha Gopalakrishna 28-June-1996
// File updated by Oliver Wallace 9-July-1996
// File updated by Soomin Kim 6-April-1998
// Copyright 1995-1997 Microsoft Corporation. All Rights Reserved.
//
=====
#include "stdafx.h"
#include "urlmon.h"
#include "wininet.h"
#include "resource.h"
#include "Prog.h"

```

```

//
=====
//                                     CBindStatusCallback Implementation
//
=====

// -----
-
// %%Function: CBindStatusCallback::CBindStatusCallback
// -----
-
CBindStatusCallback::CBindStatusCallback(HWND hwndFrame, LPCTSTR szFile)
{
    TRACE("CBindStatusCallback::CBindStatusCallback\n");
    m_hFrame          = hwndFrame;
    m_pbinding         = NULL;
    m_pstm             = NULL;
    m_cRef             = 1;
    m_pFile            = NULL;
    m_strFile          = szFile;
} // CBindStatusCallback

// -----
-
// %%Function: CBindStatusCallback::~CBindStatusCallback
// -----
-
CBindStatusCallback::~CBindStatusCallback()
{
    TRACE("CBindStatusCallback::~CBindStatusCallback\n");
} // ~CBindStatusCallback

/*
inline void CBindStatusCallback::SetWndText(HWND hwnd, LPCWSTR szText)
{
    if (IsWindow(hwnd))
    {
        char rgchBuf[INTERNET_MAX_PATH_LENGTH];
        WideCharToMultiByte(CP_ACP, 0, szText, -1, rgchBuf,
INTERNET_MAX_PATH_LENGTH, 0, 0);
        SetWindowText(hwnd, rgchBuf);
    }
}
*/

// -----
-
// %%Function: CBindStatusCallback::QueryInterface
// -----
-
STDMETHODIMP CBindStatusCallback::QueryInterface(REFIID riid, void** ppv)
{
    *ppv = NULL;

    if (riid==IID_IUnknown || riid==IID_IBindStatusCallback)
    {
        *ppv = this;
    }
}

```

```

        AddRef();
        return S_OK;
    }
    return E_NOINTERFACE;
} // CBindStatusCallback::QueryInterface

// -----
// %%Function: CBindStatusCallback::OnStartBinding
// -----
STDMETHODIMP CBindStatusCallback::OnStartBinding(DWORD dwReserved, IBinding*
pbinding)
{
    TRACE("CBindStatusCallback::OnStartBinding\n");
    if (m_pbinding)
        m_pbinding->Release();
    m_pbinding = pbinding;
    if (m_pbinding)
    {
        m_pbinding->AddRef();
        PostMessage(CMD_BIND_START); // SetStatus(L"Status: Starting to
bind...");
    }
    TRACE0("m_pFile = new CFile;\n");
    m_pFile = new CFile;
    if (!m_pFile->Open(m_strFile, CFile::modeReadWrite | CFile::modeCreate
| CFile::shareExclusive))
    {
        AfxMessageBox("Failed to open file");
        return FALSE;
    }
    PostMessage(CMD_BIND_FILE_OPENED);
    return S_OK;
} // CBindStatusCallback::OnStartBinding

// -----
// %%Function: CBindStatusCallback::OnStopBinding
// -----
STDMETHODIMP CBindStatusCallback::OnStopBinding(HRESULT hrStatus, LPCWSTR
pszError)
{
    TRACE("CBindStatusCallback::OnStopBinding\n");
    if (hrStatus)
        PostMessage(CMD_BIND_FAILED); // SetStatus(L"Status: File
download Failed.");

    if (m_pbinding)
    {
        m_pbinding->Release();
        m_pbinding = NULL;
    }

    CloseFile();
    return S_OK;
}

```

```

} // CBindStatusCallback::OnStopBinding

void CBindStatusCallback::CloseFile()
{
    if (m_pFile)
    {
        TRACE("m_pFile->Close();\n");
        m_pFile->Close();
        delete m_pFile;
        m_pFile = NULL;
        PostMessage(CMD_BIND_FILE_CLOSED);
    }
}

// -----
// %%Function: CBindStatusCallback::GetPriority
// -----
-
STDMETHODIMP CBindStatusCallback::GetPriority(LONG* pnPriority)
{
    return E_NOTIMPL;
} // CBindStatusCallback::GetPriority

// -----
// %%Function: CBindStatusCallback::OnLowResource
// -----
-
STDMETHODIMP CBindStatusCallback::OnLowResource(DWORD dwReserved)
{
    return E_NOTIMPL;
} // CBindStatusCallback::OnLowResource

// -----
// %%Function: CBindStatusCallback::OnProgress
// -----
-
STDMETHODIMP CBindStatusCallback::OnProgress(ULONG ulProgress, ULONG
ulProgressMax,
                                                    ULONG
ulStatusCode, LPCWSTR szStatusText)
{
    TRACE("CBindStatusCallback::OnProgress(%lu)\n", ulProgress);
    char sz[255];

    if (szStatusText)
        WideCharToMultiByte(CP_ACP, 0, szStatusText, -1, sz, 255, 0, 0);

    char msg[256];
    // WCHAR out[256*2];

    wsprintf(msg, "Progress: %s %d of %d ", sz, ulProgress, (ulProgress >
ulProgressMax) ? ulProgress : ulProgressMax);
    // MultiByteToWideChar(CP_ACP, 0, msg, -1, out, sizeof(out));

```



```

        SendMessage(CMD_BIND_PROGRESS_MSG, 0, (LPARAM)msg);
        TRACE("%lu, %lu\n", ulProgress, ulProgressMax);
        SendMessage(CMD_BIND_PROGRESS_BAR, (WPARAM)ulProgress,
(LPARAM)ulProgressMax); // by value
        return NOERROR;
    } // CBindStatusCallback::OnProgress

// -----
// %%Function: CBindStatusCallback::GetBindInfo
// -----
STDMETHODIMP CBindStatusCallback::GetBindInfo(DWORD* pgrfBINDF, BINDINFO*
pbindInfo)
{
    if (!pbindInfo || !pbindInfo->cbSize || !pgrfBINDF)
        return E_INVALIDARG;

    *pgrfBINDF = BINDF_ASYNCHRONOUS | BINDF_ASYNCSTORAGE | BINDF_PULLDATA |
        BINDF_GETNEWESTVERSION | BINDF_NOWRITECACHE;

    // remember incoming cbSize
    ULONG cbSize = pbindInfo->cbSize;
    // zero out structure
    ::ZeroMemory(pbindInfo, cbSize);

    // restore cbSize
    pbindInfo->cbSize = cbSize;
    pbindInfo->dwBindVerb = BINDVERB_GET;

    return S_OK;
} // CBindStatusCallback::GetBindInfo

// -----
// %%Function: CBindStatusCallback::OnDataAvailable
// -----
STDMETHODIMP CBindStatusCallback::OnDataAvailable(DWORD grfBSCF, DWORD
dwSize, FORMATETC* pfmtetc, STGMEDIUM* pstgmed)
{
    TRACE("CBindStatusCallback::OnDataAvailable(%ld)\n", dwSize);
    HRESULT hr=S_OK;

    // Get the Stream passed
    if (BSCF_FIRSTDATANOTIFICATION & grfBSCF)
    {
        if (!m_pstm && (pstgmed->tymed == TYMED_ISTREAM))
        {
            m_pstm = pstgmed->pstm;
            if (m_pstm)
                m_pstm->AddRef();
        }
    }

    // If there is some data to be read then go ahead and read them
    if (m_pstm && (dwSize > 0))

```

```

    {
        DWORD dwRead = dwSize;           // Minimum amount available that
hasn't been read
        DWORD dwActuallyRead = 0;        // Placeholder for amount read
during this pull

        if (dwRead > 0)
        {
            do
            {
                char* pNewstr = new char[dwRead + 1];
                if (!pNewstr)
                    return S_FALSE;
                hr = m_pstm->Read(pNewstr, dwRead, &dwActuallyRead);
                pNewstr[dwActuallyRead] = 0;
                // If we really read something then lets add it to
the Edit box

                if (dwActuallyRead > 0)
                {
                    TRY
                    {
                        m_pFile->Write(pNewstr, dwActuallyRead);
                    }
                    CATCH(CFileException, e)
                    {
                        TRACE0("Failed to write file.\n");
                        return FALSE;
                    }
                    END_CATCH
                }
                delete [] pNewstr;
            } while (!(hr == E_PENDING || hr == S_FALSE) &&
SUCCEEDED(hr));
        }
        // if (m_pstm && dwSize > 0)

        if (BSCF_LASTDATANOTIFICATION & grfBSCF)
        {
            if (m_pstm)
                m_pstm->Release();
            hr = S_OK; // If it was the last data then we should return S_OK
as we just finished reading everything
            PostMessage(CMD_BIND_DOWNLOAD_DONE); // SetStatus(L"Status:
File downloaded.");
            TRACE0("Status: File downloaded.\n");
        }
        return hr;
    } // CBindStatusCallback::OnDataAvailable

// -----
// %%Function: CBindStatusCallback::OnObjectAvailable
// -----
STDMETHODIMP CBindStatusCallback::OnObjectAvailable(REFIID riid, IUnknown*
punk)
{
    return E_NOTIMPL;
}

```

```

} // CBindStatusCallback::OnObjectAvailable

//
=====
//                                     CDownload Implementation
//
=====

// -----
-
// %%Function: CDownload::CDownload
// -----
-
CDownload::CDownload(LPCWSTR szURL)
{
    TRACE("CDownload::CDownload\n");
    m_url = szURL;
    m_pmk = 0;
    m_pbc = 0;
    m_pbsc = 0;
} // CDownload

// -----
-
// %%Function: CDownload::~CDownload
// -----
-
CDownload::~CDownload()
{
    TRACE("CDownload::~CDownload\n");
    if (m_pmk)
        m_pmk->Release();
    if (m_pbc)
        m_pbc->Release();
    if (m_pbsc)
    {
        m_pbsc->Release();
        delete m_pbsc;
    }
} // ~CDownload

// -----
-
// %%Function: CDownload::DoDownload
// -----
-
HRESULT CDownload::DoDownload(HWND hwndFrame, LPCTSTR szFile)
{
    TRACE("CDownload::DoDownload\n");
    IStream* pstm = NULL;

    HRESULT hr = CreateURLMoniker(NULL, m_url, &m_pmk);
    if (FAILED(hr))
        goto LErrExit;

    m_pbsc = new CBindStatusCallback(hwndFrame, szFile);
    if (!m_pbsc)

```

```

    {
        hr = E_OUTOFMEMORY;
        goto LErrExit;
    }

    hr = CreateBindCtx(0, &m_pbc);
    if (FAILED(hr))
        goto LErrExit;

    hr = RegisterBindStatusCallback(m_pbc, m_pbsc, 0, 0L);
    if (FAILED(hr))
        goto LErrExit;

    hr = m_pmk->BindToStorage(m_pbc, 0, IID_IStream, (void**)&pstm);
    if (FAILED(hr))
        goto LErrExit;

    return hr;

LErrExit:
    if (m_pbc)
    {
        m_pbc->Release();
        m_pbc = NULL;
    }
    if (m_pbsc)
    {
        m_pbsc->Release();
        m_pbsc = NULL;
    }
    if (m_pmk)
    {
        m_pmk->Release();
        m_pmk = NULL;
    }
    if (pstm)
    {
        pstm->Release();
        pstm = NULL;
    }
    return hr;
} // CDownload::DoDownload

void CDownload::CancelDownload()
{
    if (m_pbsc)
    {
        m_pbsc->CloseFile();
    }
}

```

```

#ifndef __PROG_H
#define __PROG_H

#include "urlmon.h"
#include "wininet.h"

#define BASE (WM_USER + 200 + 100)
const UINT CMD_BIND_START = BASE;
const UINT CMD_BIND_FAILED = BASE+1;
const UINT CMD_BIND_FILE_OPENED = BASE+2;
const UINT CMD_BIND_FILE_CLOSED = BASE+3;
const UINT CMD_BIND_PROGRESS_MSG = BASE+4; //
SendMessage(CMD_BIND_PROGRESS_MSG, 0, (LPARAM)msg);
const UINT CMD_BIND_PROGRESS_BAR = BASE+5; //
SendMessage(CMD_BIND_PROGRESS_BAR, (LPARAM)ulProgress,
(LPARAM)ulProgressMax); // by value
const UINT CMD_BIND_DOWNLOAD_DONE = BASE+6;
#undef BASE

// %%Classes: -----
-
class CBindStatusCallback : public IBindStatusCallback
{
public:
    // IUnknown methods
    STDMETHODIMP QueryInterface(REFIID riid, void **ppv);
    STDMETHODIMP_(ULONG) AddRef() { return m_cRef++; }
    STDMETHODIMP_(ULONG) Release() { return m_cRef--; }
    //if (--m_cRef == 0) { delete this; return 0; } return m_cRef; }

    // IBindStatusCallback methods
    STDMETHODIMP OnStartBinding(DWORD dwReserved, IBinding* pbinding);
    STDMETHODIMP GetPriority(LONG* pnPriority);
    STDMETHODIMP OnLowResource(DWORD dwReserved);
    STDMETHODIMP OnProgress(ULONG ulProgress, ULONG ulProgressMax, ULONG
ulStatusCode,
                                LPCWSTR pwzStatusText);
    STDMETHODIMP OnStopBinding(HRESULT hrResult, LPCWSTR szError);
    STDMETHODIMP GetBindInfo(DWORD* pgrfBINDF, BINDINFO* pbindinfo);
    STDMETHODIMP OnDataAvailable(DWORD grfBSCF, DWORD dwSize, FORMATETC
*pfmtetc,
                                STGMEDIUM* pstgmed);
    STDMETHODIMP OnObjectAvailable(REFIID riid, IUnknown* punk);

    // constructors/destructors
    CBindStatusCallback(HWND hwndFrame, LPCTSTR szFile);
    ~CBindStatusCallback();

    BOOL PostMessage(UINT message, WPARAM wParam=0L, LPARAM lParam=0L)
const
    { return ::PostMessage(m_hFrame, message, wParam, lParam); }
}

    BOOL SendMessage(UINT message, WPARAM wParam=0L, LPARAM lParam=0L)
const
    { return ::SendMessage(m_hFrame, message, wParam, lParam); }
}

    void CloseFile();

```

```

        // data members
        DWORD      m_cRef;
        IBinding*   m_pbinding;
        IStream*    m_pstm;
        HWND        m_hFrame;
        CFile*       m_pFile;
        CString     m_strFile;
};

class CDownload
{
public:
    CDownload(LPCWSTR szURL);
    ~CDownload();
    HRESULT      DoDownload(HWND hwndFrame, LPCTSTR szFile);
    void         CancelDownload();
    LPCWSTR      m_url;

private:
    IMoniker*     m_pmk;
    IBindCtx*     m_pbc;
    CBindStatusCallback* m_pbsc;
};

#endif // __PROG_H

```

```

// Progress.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Progress.h"
#include "ProgressDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CProgressApp

BEGIN_MESSAGE_MAP(CProgressApp, CWinApp)
    //{{AFX_MSG_MAP(CProgressApp)
    // NOTE - the ClassWizard will add and remove mapping macros
    here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

////////////////////////////////////
// CProgressApp construction

CProgressApp::CProgressApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance

    //////////////////////////////////////
    // The one and only CProgressApp object

    CProgressApp theApp;

    //////////////////////////////////////
    // CProgressApp initialization

    BOOL CProgressApp::InitInstance()
    {
        if (!AfxSocketInit())
        {
            AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
            return FALSE;
        }

        AfxEnableControlContainer();

        // Standard initialization
        // If you are not using these features and wish to reduce the size
        // of your final executable, you should remove from the following
        // the specific initialization routines you do not need.

```

```

#ifdef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC
statically
#endif

    CProgressDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    // application, rather than start the application's message pump.
    return FALSE;
}

```



```

// Progress.h : main header file for the PROGRESS application
//

#ifndef AFX_PROGRESS_H_C8A475E7_CD64_11D1_80E2_080009B9F339__INCLUDED_
#define AFX_PROGRESS_H_C8A475E7_CD64_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CProgressApp:
// See Progress.cpp for the implementation of this class
//

class CProgressApp : public CWinApp
{
public:
    CProgressApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CProgressApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CProgressApp)
    // NOTE - the ClassWizard will add and remove member functions
    here.
    // DO NOT EDIT what you see in these blocks of generated code
    !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROGRESS_H_C8A475E7_CD64_11D1_80E2_080009B9F339__INCLUDED_

```

```

// ProgressDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Progress.h"
#include "ProgressDlg.h"
#include "Prog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CProgressDlg dialog

CProgressDlg::CProgressDlg(CWnd* pParent /*=NULL*/)
: CDialog(CProgressDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CProgressDlg)
    m_strDisplay = _T("");
    m_strProgress = _T("");
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in
Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_pDownload = NULL;
    m_aDI = NULL;
    m_nDis = 0;
}

CProgressDlg::~CProgressDlg()
{
    if (m_pDownload)
        delete m_pDownload;
    if (m_aDI)
        delete [] m_aDI;

    // CoUninitialize();
}

void CProgressDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CProgressDlg)
    DDX_Control(pDX, IDC_PROGRESSBAR, m_pbProgress);
    DDX_Text(pDX, IDC_DISPLAY, m_strDisplay);
    DDX_Text(pDX, IDC_PROGRESS, m_strProgress);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CProgressDlg, CDialog)
    //{{AFX_MSG_MAP(CProgressDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()

```

```

        ///}AFX_MSG_MAP
        ON_MESSAGE(CMD_BIND_START,          OnBindStart)
        ON_MESSAGE(CMD_BIND_FAILED,         OnBindFailed)
        ON_MESSAGE(CMD_BIND_FILE_OPENED,    OnBindFileOpened)
        ON_MESSAGE(CMD_BIND_FILE_CLOSED,    OnBindFileClosed)
        ON_MESSAGE(CMD_BIND_PROGRESS_MSG,   OnBindProgressMsg)      //
        SendMessage(CMD_BIND_PROGRESS_MSG, 0, (LPARAM)msg);
        ON_MESSAGE(CMD_BIND_PROGRESS_BAR,   OnBindProgressBar)     //
        SendMessage(CMD_BIND_PROGRESS_BAR, (WPARAM)ulProgress,
        (LPARAM)ulProgressMax);
        ON_MESSAGE(CMD_BIND_DOWNLOAD_DONE,  OnBindDownloadDone)
    END_MESSAGE_MAP()

    //////////////////////////////////////
    // CProgressDlg message handlers

    BOOL CProgressDlg::OnInitDialog()
    {
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system commandrange.
        ASSERT((IDM_ABOUTBOX & 0xFFFF) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
            CString strAboutMenu;
            strAboutMenu.LoadString(IDS_ABOUTBOX);
            if (!strAboutMenu.IsEmpty())
            {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
            }

            // Set the icon for this dialog. The framework does this automatically
            // when the application's main window is not a dialog
            SetIcon(m_hIcon, TRUE);          // Set big icon
            SetIcon(m_hIcon, FALSE);         // Set small icon

            // HRESULT hr = CoInitialize(NULL);
            // if (FAILED(hr))
            //     return FALSE;

            return TRUE; // return TRUE unless you set the focus to a control
        }

    void CProgressDlg::OnSysCommand(UINT nID, LPARAM lParam)
    {
        // if ((nID & 0xFFFF) == IDM_ABOUTBOX)
        // {
        //     CAboutDlg dlgAbout;
        //     dlgAbout.DoModal();
        // }
    }

```

```

//      }
//      else
//      {
//          CDialog::OnSysCommand(nID, lParam);
//      }
//  }

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CProgressDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CProgressDlg::OnQueryDragIcon()
{
    return (HCURSOR)m_hIcon;
}

void CProgressDlg::OnOK()
{
    CString strURL("ftp://88.1.26.2/UniChat/u2files.txt"); //
    203.241.132.83
    static WCHAR szDefaultURL[MAX_PATH]; //
    L"http://www.msn.com";
    MultiByteToWideChar(CP_ACP, 0, strURL, -1, szDefaultURL, MAX_PATH);
    if (m_pDownload)
    {
        delete m_pDownload;
    }
    m_pDownload = new CDownload(szDefaultURL);

    GetDlgItem(IDOK)->EnableWindow(FALSE);
}

```

```

        GetDlgItem(IDCANCEL) ->EnableWindow(FALSE);

        char rgchBuf[INTERNET_MAX_PATH_LENGTH];
        WideCharToMultiByte(CP_ACP, 0, m_pDownload->m_url, -1, rgchBuf,
MAX_PATH, 0, 0);
        SetWindowText(rgchBuf);

        m_strDisplay = "Status: Initiating Bind...";
        UpdateData(FALSE);
        m_pDownload->DoDownload(GetSafeHwnd(), "test.txt");
        GetDlgItem(IDCANCEL) ->EnableWindow(TRUE);

//    CDialog::OnOK();
}

void CProgressDlg::OnCancel()
{
    if (m_pDownload)
        m_pDownload->CancelDownload();
//    EndDialog(hwndDlg, 0);
    CDialog::OnCancel();
}

LRESULT CProgressDlg::OnBindStart(WPARAM, LPARAM)
{
    m_strDisplay = "Status: Preparing download..."; //Starting to bind...";
    UpdateData(FALSE);    // Write
    return 0;
}

LRESULT CProgressDlg::OnBindFailed(WPARAM, LPARAM)
{
    m_strDisplay = "Status: File download Failed.";
    UpdateData(FALSE);    // Write
    return 0;
}

LRESULT CProgressDlg::OnBindFileOpened(WPARAM, LPARAM)
{
    return 0;
}

LRESULT CProgressDlg::OnBindFileClosed(WPARAM, LPARAM)
{
    return 0;
}

LRESULT CProgressDlg::OnBindProgressMsg(WPARAM wParam, LPARAM lParam)
{
    char* szMsg = (char*)lParam;
    if (szMsg && strlen(szMsg))
    {
        m_strProgress = szMsg;
        UpdateData(FALSE);    // Write
    }
    return 0;
}

```

```
LRESULT CProgressDlg::OnBindProgressBar(WPARAM wParam, LPARAM lParam)
{
    ULONG cProgress      = (ULONG)wParam;
    ULONG maxProgress = (ULONG)lParam;
    m_pbProgress.SetRange(0, 100);
    m_pbProgress.SetPos(maxProgress ? cProgress * 100 / maxProgress : 0);
    return 0;
}

LRESULT CProgressDlg::OnBindDownloadDone(WPARAM, LPARAM)
{
    m_strDisplay = "Status: File downloaded.";
    UpdateData(FALSE); // Write
    GetDlgItem(IDOK) ->EnableWindow(TRUE);
    return 0;
}
```

Progress\ProgressDlg.h

```
// ProgressDlg.h : header file
//
```

```
#if
!defined(AFX_PROGRESSDLG_H__C8A475E9_CD64_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_PROGRESSDLG_H__C8A475E9_CD64_11D1_80E2_080009B9F339__INCLUDED_
```

```
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
////////////////////////////////////
// CProgressDlg dialog
class CDownload;
```

```
class CDownInfo
{
public:    // CFileStatus
    CString      m_strFile;
    CTime        m_mtime;    // The date and time the file was created.
    LONG         m_size;      // The logical size of the file in bytes
    BOOL         m_bExtract;
};
```

```
class CProgressDlg : public CDialog
{
// Construction
public:
    CProgressDlg(CWnd* pParent = NULL); // standard constructor
    ~CProgressDlg();
```

```
// Dialog Data
//{{AFX_DATA(CProgressDlg)
enum { IDD = IDD_PROGRESS_DIALOG };
CProgressCtrl      m_pbProgress;
CString            m_strDisplay;
CString            m_strProgress;
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CProgressDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
//}}AFX_VIRTUAL
```

```
// Implementation
protected:
    HICON          m_hIcon;
    CDownload*     m_pDownload;
    CDownInfo*     m_aDI;
    int            m_nDis;

// ON_MESSAGE
LRESULT OnBindStart(WPARAM, LPARAM);
LRESULT OnBindFailed(WPARAM, LPARAM);
LRESULT OnBindFileOpened(WPARAM, LPARAM);
```

```
LRESULT OnBindFileClosed(WPARAM, LPARAM);
LRESULT OnBindProgressMsg(WPARAM, LPARAM);
LRESULT OnBindProgressBar(WPARAM, LPARAM);
LRESULT OnBindDownloadDone(WPARAM, LPARAM);

// Generated message map functions
//{{AFX_MSG(CProgressDlg)
virtual BOOL OnInitDialog();
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
virtual void OnOK();
virtual void OnCancel();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
#ifndef AFX_PROGRESSDLG_H__C8A475E9_CD64_11D1_80E2_080009B9F339__INCLUDED_
```



```
=====
MICROSOFT FOUNDATION CLASS LIBRARY : Progress
=====
```

AppWizard has created this Progress application for you. This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that make up your Progress application.

Progress.h

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CProgressApp application class.

Progress.cpp

This is the main application source file that contains the application class CProgressApp.

Progress.rc

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

res\Progress.ico

This is an icon file, which is used as the application's icon. This icon is included by the main resource file Progress.rc.

res\Progress.rc2

This file contains resources that are not edited by Microsoft Developer Studio. You should place all resources not editable by the resource editor in this file.

Progress.clw

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

```
////////////////////////////////////////////////////////////////
```

AppWizard creates one dialog class:

ProgressDlg.h, ProgressDlg.cpp - the dialog

These files contain your CProgressDlg class. This class defines the behavior of your application's main dialog. The dialog's template is in Progress.rc, which can be edited in Microsoft Developer Studio.

```
////////////////////////////////////////////////////////////////
```

Other standard files:

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header (PCH) file named Progress.pch and a precompiled types file named StdAfx.obj.

Resource.h

This is the standard header file, which defines new resource IDs. Microsoft Developer Studio reads and updates this file.

Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you should add to or customize.

If your application uses MFC in a shared DLL, and your application is in a language other than the operating system's current language, you will need to copy the corresponding localized resources MFC40XXX.DLL from the Microsoft Visual C++ CD-ROM onto the system or system32 directory, and rename it to be MFCLOC.DLL. ("XXX" stands for the language abbreviation. For example, MFC40DEU.DLL contains resources translated to German.) If you don't do this, some of the UI elements of your application will remain in the language of the operating system.

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Progress.rc
//
#define IDM_ABOUTBOX                0x0010
#define IDD_ABOUTBOX                100
#define IDS_ABOUTBOX                101
#define IDD_PROGRESS_DIALOG        102
#define IDP_SOCKETS_INIT_FAILED    103
#define IDR_MAINFRAME              128
#define IDC_EDIT_MSG               1000
#define IDC_DISPLAY                1001
#define IDC_PROGRESS               1002
#define IDC_PROGRESSBAR            1013

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE    129
#define _APS_NEXT_COMMAND_VALUE    32771
#define _APS_NEXT_CONTROL_VALUE    1001
#define _APS_NEXT_SYMED_VALUE      101
#endif
#endif
```

Progress\StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//     Progress.pch will be the pre-compiled header
//     stdafx.obj will contain the pre-compiled type information
```

```
#include "stdafx.h"
```

```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H__C8A475EB_CD64_11D1_80E2_080009B9F339__INCLUDED_)
#define AFX_STDAFX_H__C8A475EB_CD64_11D1_80E2_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h> // MFC core and standard components
#include <afxext.h> // MFC extensions
#include <afxdisp.h> // MFC OLE automation classes
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h> // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxsock.h> // MFC socket extensions

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__C8A475EB_CD64_11D1_80E2_080009B9F339__INCLUDED_)

```

```
#FILELIST=
{
u2res000.rit=0,4614,1998,4,1,10,22; // 0: do not extract
0000csin.sit=0,1811,1998,3,31,14,41;
0001ctrm.sit=0,2649,1998,4,1,11,10;
0001demo.sit=0,2454,1998,3,26,14,59;
0002ctrm.sit=0,1955,1998,4,1,11,10;
0002demo.sit=0,1851,1998,3,26,14,59;
0003ctrm.sit=0,2256,1998,4,1,11,11;
0003demo.sit=0,2046,1998,3,6,15,0;
0004ctrm.sit=0,2075,1998,4,1,11,12;
0005ctrm.sit=0,2368,1998,4,1,11,13;
0010csin.sit=0,1781,1998,4,1,8,31;
0020csin.sit=0,1882,1998,3,23,15,5;
1000csin.sit=0,1909,1998,3,24,20,24;
1010csin.sit=0,1802,1998,3,24,15,33;
1020csin.sit=0,1531,1998,3,24,16,40;
2000csin.sit=0,1997,1998,3,24,21,7;
2010csin.sit=0,2038,1998,3,31,13,3;
2020csin.sit=0,2383,1998,3,24,21,16;
3000csin.sit=0,2289,1998,3,25,15,44;
3010csin.sit=0,1800,1998,3,26,15,26;
3020csin.sit=0,1980,1998,3,26,12,50;
4000csin.sit=0,2290,1998,3,26,22,55;
4010csin.sit=0,1918,1998,3,27,20,25;
4020csin.sit=0,2087,1998,3,30,16,44;
cg00.uds=1,149701,1998,3,7,17,36; // 1: extract
}
```

```

//
//  CBubbleListNotifyObj :
//
//  (C) Programmed by Kim,
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "dib.h"
#include "dibpal.h"
#include "bubbleno.h"
#include "bblstno.h"
#include "bubble.h"
#include "bubblst.h"
#include "osbview.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CBubbleListNotifyObj

CBubbleListNotifyObj::CBubbleListNotifyObj()
{
    m_pBubbleList = NULL;
    m_pBufferView = NULL;
}

CBubbleListNotifyObj::~CBubbleListNotifyObj()
{
}

// Notification callback from a CBubble object.
void CBubbleListNotifyObj::Change(CBubble* pBubble, CHANGETYPE change,
                                   CRect* pRect1, CRect* pRect2)
{
    if (change & CBubbleNotifyObj::ZORDER)
    {
        // Reposition the bubble in the z-order list.
        ASSERT(m_pBubbleList);
        m_pBubbleList->Reorder(pBubble);
        // Add the bubble position to the dirty list.
        ASSERT(m_pBufferView);
        m_pBufferView->AddDirtyRegion(pRect1);
    }
    if (change & CBubbleNotifyObj::POSITION)
    {
        // pRect1 and pRect2 point to old and new rectangle positions;
        // add these rectangles to the dirty list.
        ASSERT(m_pBufferView);
        m_pBufferView->AddDirtyRegion(pRect1);
        m_pBufferView->AddDirtyRegion(pRect2);
    }
}

```

```

    if (change & CBubbleNotifyObj::IMAGE)
    {
        // redraw the bubble
        // Add the bubble position to the dirty list.
        ASSERT(m_pBufferView);
        m_pBufferView->AddDirtyRegion(pRect1);
    }
}

```



```

//
//  CBubbleListNotifyObj:
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
//  This is a class derived from CBubbleNotifyObj which is used in
//  CBubbleList to handle notification calls from CBubble objects.
//

#ifndef __BBLSTNO__
#define __BBLSTNO__

#include "BubbleNo.h"
class CBubbleList;
class COSBView;

class AFX_EXT_CLASS CBubbleListNotifyObj : public CBubbleNotifyObj
{
public:
    CBubbleListNotifyObj();
    ~CBubbleListNotifyObj();
    void SetList(CBubbleList* pBubbleList)    {m_pBubbleList =
pBubbleList;}
    void SetView(COSBView* pBufferView)      {m_pBufferView =
pBufferView;}
    void Change(CBubble* pBubble, CHANGETYPE change,
                CRect* pRect1=NULL, CRect* pRect2=NULL);

protected:
    CBubbleList*    m_pBubbleList;
    COSBView*      m_pBufferView;
};
#endif // __BBLSTNO__

```

```

//
//  CBubble :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "BubbleNo.h"
#include "Bubble.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

const DWORD LONGTIMELATER = 3600000L;      // 1 hour later

////////////////////////////////////
// CBubble

IMPLEMENT_SERIAL(CBubble, CObject, 0 /* Schema number */)

CBubble::CBubble()
{
    m_bShow = FALSE;
    m_z      = 0;
    m_rcBB.SetRectEmpty();
    m_rcDT.SetRectEmpty();
    m_strText.Empty();
    m_pNotifyObj = NULL;
    // later change by using property page
    m_RREllipse = CPoint(12, 12); // 16, 16);
    m_maxWidth  = 100; // 120;
    m_TA        = TA_LEFT;
    m_textcolor = PALETTE_RGB(0, 0, 0);
    m_brush.CreateSolidBrush(PALETTE_RGB(255, 235, 187));
    // m_brush.CreateHatchBrush(HS_DIAGCROSS, RGB(255, 255, 128));
    // m_brush.CreateStockObject(NULL_BRUSH);
    m_font.CreateFont(-12, 0, 0, 0, FW_NORMAL,
                     FALSE, FALSE, 0, // bItalic, bUnderline, cStrikeOut
                     DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
                     DEFAULT_PITCH || FF_DONTCARE,
#ifdef _KOREAN
                     "±¼.²Ã¼"); // ¹ÜÄÄÃ¼, ±¼.²Ã¼, µ.¿òÃ¼
#else
                     "Times New Roman");
#endif
    m_dwAlarmTick = ::GetTickCount() + LONGTIMELATER;
}

CBubble::~CBubble()
{
}

```

```

////////////////////////////////////
// CBubble serialization

```

```

void CBubble::Serialize(CArchive& ar)

```

```

{
    CObject::Serialize(ar);
    if (ar.IsStoring())
    {
        ar << (DWORD)m_z;
        ar << m_rcBB;
        ar << m_strText;
        ar << m_RREllipse;
        ar << (DWORD)m_maxWidth;
        ar << m_textcolor;
    }
    else
    {
        DWORD dw;
        ar >> dw; m_z = (int)dw;
        ar >> m_rcBB;
        ar >> m_strText;
        ar >> m_RREllipse;
        ar >> dw; m_maxWidth = (int)dw;
        ar >> m_textcolor;
    }
}

```

```

////////////////////////////////////
// CBubble commands

```

```

void CBubble::SetKind(const int n)

```

```

{
    switch (n)
    {
        case 0: // Yellow
            if (HBRUSH(m_brush))
                m_brush.DeleteObject();
            m_brush.CreateSolidBrush(PALETTERGB(255,255,195));
            //255,239,191
            break;
        case 1: // White
            if (HBRUSH(m_brush))
                m_brush.DeleteObject();
            m_brush.CreateSolidBrush(PALETTERGB(222,231,231));
            break;
        case 2: // Green
            if (HBRUSH(m_brush))
                m_brush.DeleteObject();
            m_brush.CreateSolidBrush(PALETTERGB(206,255,90));
            break;
    }
}

```

```

void CBubble::Render(const HBITMAP hbm, CPalette* pPal, const CRect*
pClipRect)
{

```

```

    if (!m_bShow)
        return;
    if (m_strText.IsEmpty())
        return;

    CDC dcMem;
    dcMem.CreateCompatibleDC(NULL);    // create a memory dc that is
compatible with current screen
    CBitmap* pbmOld = dcMem.SelectObject(CBitmap::FromHandle(hbm));
    // If a clip rectangle was supplied, see if the bubble is visible
inside the rectangle.
    if (pClipRect)
    {
        CRect rcDraw;
        if (!rcDraw.IntersectRect(pClipRect, &m_rcBBT))
            return; // not visible
    }

/*
    CPalette* ppalOld = NULL;
    if (pPal)
    {
        ppalOld = dcMem.SelectPalette(pPal, FALSE);    // foreground
        dcMem.RealizePalette();
    }

*/
    CBrush* brOld = dcMem.SelectObject(&m_brush);
    dcMem.SetBkMode(TRANSPARENT);
    dcMem.SetTextColor(m_textcolor);

    dcMem.RoundRect(&m_rcBB, m_RREllipse);

    static POINT aP[7];
    aP[6].y = aP[0].y = m_rcBB.bottom - 1;
    aP[5].y = aP[1].y = aP[0].y + 2;
    aP[4].y = aP[2].y = aP[1].y + 2;
    aP[3].y = aP[2].y + 1;
    if (m_TA == TA_RIGHT)    // shape ))
    {
        aP[0].x = m_rcBB.right - 15;
        aP[1].x = aP[0].x - 1;
        aP[2].x = aP[1].x - 2;
        aP[3].x = aP[2].x - 1;
        aP[4].x = aP[3].x + 3;
        aP[5].x = aP[4].x + 4;
        aP[6].x = aP[5].x + 2;
    }
    else // TA_LEFT shape ((
    {
        aP[0].x = m_rcBB.left + 15;
        aP[1].x = aP[0].x + 1;
        aP[2].x = aP[1].x + 2;
        aP[3].x = aP[2].x + 1;
        aP[4].x = aP[3].x - 3;
        aP[5].x = aP[4].x - 4;
        aP[6].x = aP[5].x - 2;
    }
    CPen* penOld = (CPen*)dcMem.SelectStockObject(NULL_PEN);

```

```

        dcMem.Polygon(aP, sizeof(aP)/sizeof(aP[0])); // draw Polygon (2D)
without border
        dcMem.SelectObject(penOld);
        dcMem.Polyline(aP, sizeof(aP)/sizeof(aP[0])); // draw border with
default pen

        CFont* pOldFont = dcMem.SelectObject(&m_font);
        dcMem.DrawText(m_strText, -1, &m_rcDT, DT_LEFT | DT_WORDBREAK);
        dcMem.SelectObject(pOldFont);
// if (ppalOld)
//     dcMem.SelectPalette(ppalOld, FALSE);
// GdiFlush();
        dcMem.SelectObject(brOld);
        dcMem.SelectObject(pbmOld);
    }

void CBubble::Show(const BOOL bShow)
{
    m_bShow = bShow ? TRUE : FALSE;
    if (m_pNotifyObj)
        m_pNotifyObj->Change(this, CBubbleNotifyObj::IMAGE, &m_rcBBT);
}

// Test for a hit in a non-transparent area
BOOL CBubble::HitTest(const CPoint& point) const
{
    // Test if the point is inside the sprite rectangle
    if (m_rcBB.PtInRect(point))
        return TRUE; // hit
    return FALSE;
}

// Set a new Z-order.
void CBubble::SetZ(const int z)
{
    if (m_z == z)
        return;
    m_z = z;
    // See if we have to notify anyone.
    if (m_pNotifyObj)
        m_pNotifyObj->Change(this, CBubbleNotifyObj::ZORDER, &m_rcBBT);
}

// call this function before SetLT, lr=left or right according to ta
void CBubble::TextOut(const int lr, const int bottom, const CString& str)
{
    // Save the current position.
    CRect rcOld = m_rcBBT;
    m_strText = str; // Although str is a NULL, that means she said
nothing...
    // Move to new position.
    m_rcDT.bottom = bottom;
    m_rcDT.top = m_rcDT.bottom;
    if (m_TA == TA_LEFT)
    {
        m_rcDT.left = lr;
    }
}

```

```

        m_rcDT.right      = m_rcDT.left + m_maxWidth;    // specify
maximum width
    }
    else
    {
        m_rcDT.right      = lr;
        m_rcDT.left       = m_rcDT.right - m_maxWidth;
    }
    CDC dcMem;
    dcMem.CreateCompatibleDC(NULL);    // create a memory dc that is
compatible with current screen
    CFont* pOldFont = dcMem.SelectObject(&m_font);
    dcMem.DrawText(m_strText, -1, &m_rcDT, DT_CALCRECT | DT_WORDBREAK);
    // only left-aligns m_rcDT
    dcMem.SelectObject(pOldFont);

    // adjust for too small bubbles
    if (m_rcDT.Width() < (m_maxWidth / 2))
    {
        if (m_TA == TA_LEFT)
            m_rcDT.right = m_rcDT.left + m_maxWidth / 2;
        else
            m_rcDT.left = m_rcDT.right - m_maxWidth / 2;
    }
    // since DrawText only left-aligns m_rcDT, we need to right-align
    if (m_TA == TA_RIGHT)
    {
        int w = m_rcDT.Width(); // save width
        m_rcDT.right = lr;
        m_rcDT.left = m_rcDT.right - w;
    }
    m_rcDT.OffsetRect(0, -m_rcDT.Height());

    SetRectangles();

    // Notify that we have moved from our old position to our new position.
    if (m_pNotifyObj)
        m_pNotifyObj->Change(this, CBubbleNotifyObj::POSITION, &rcOld,
&m_rcBBT);
    }

void CBubble::SetRectangles()
{
    // Adjust for bubbles outside screen area
    if (m_rcDT.left < 0)
        m_rcDT.OffsetRect(-m_rcDT.left, 0);
    if (m_rcDT.top < 0)
        m_rcDT.OffsetRect(0, -m_rcDT.top);

    m_rcBB = m_rcDT;
    m_rcBB.InflateRect(m_RREllipse.x/2, m_RREllipse.y/2);

    m_rcBBT = m_rcBB;
    m_rcBBT.bottom += 8;    // BUBBLE_TAIL
}

```

```
// left-bottom position
void CBubble::MovePosition(const int lr, const int bottom)
{
    CRect rcOld = m_rcBBT;
    int dx = (m_TA == TA_LEFT) ? lr - m_rcDT.left : lr - m_rcDT.right;
    int dy = bottom - m_rcDT.bottom;

    m_rcDT.OffsetRect(dx, dy);

    SetRectangles();

    // Notify that we have moved from our old position to our new position.
    if (m_pNotifyObj)
        m_pNotifyObj->Change(this, CBubbleNotifyObj::POSITION, &rcOld,
&m_rcBBT);
}

void CBubble::SetAlarmTick(const DWORD dwAlarm)
{
    m_dwAlarmTick = dwAlarm ? dwAlarm : ::GetTickCount();
}
```

```

//
//  CBubble:
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#ifndef __BUBBLE_H
#define __BUBBLE_H

class CBubbleNotifyObj;

class AFX_EXT_CLASS CBubble : public CObject
{
    DECLARE_SERIAL(CBubble)
public:
    CBubble();
    ~CBubble();

// Attributes
    int          GetZ() const           { return m_z; }    //
Get z-order.
    int          GetWidth() const       { return m_rcBB.Width(); }
    int          GetHeight() const      { return m_rcBB.Height(); }
    void GetRect(CRect& rect) const      { rect = m_rcBBT; }
    UINT GetTextAlign() const           { return m_TA; }
    DWORD GetAlarmTick() const          { return m_dwAlarmTick; }

// Operations
    void Show(const BOOL bShow);
    BOOL IsShown() const                { return m_bShow; }
    void SetKind(const int n);
    void Serialize(CArchive& ar);
    void Render(const HBITMAP hbm, CPalette* pPal, const CRect* pClipRect
= NULL);
    BOOL HitTest(const CPoint& point) const;
    void SetZ(const int z);
    void SetTextAlign(const UINT ta)    { m_TA = ta; }
    void TextOut(const int lr, const int bottom, const CString& str);
    void MovePosition(const int lr, const int bottom);
    void SetNotificationObject(CBubbleNotifyObj* pNO) { m_pNotifyObj =
pNO; }
    void SetAlarmTick(const DWORD dwAlarm);

protected:
    void SetRectangles();

    int          m_z;                // Z-order for this bubble
    CRect        m_rcBB;              // rectangle surrounds this bubble
    CRect        m_rcBBT;             // rectangle including bubble's tail
    CRect        m_rcDT;              // used for DrawText
    CString      m_strText;
    CBubbleNotifyObj* m_pNotifyObj;    // Pointer to a notification object

private:

```


UC2Ani\Bubble.h

```

        CPoint          m_RREllipse;          // width, height of the ellipse for
the CDC::RoundRect
        int             m_maxWidth;          // maximum width for this bubble
(only width matters)
        COLORREF        m_textcolor;
        CFont           m_font;
        CBrush          m_brush;
        UINT            m_TA;                // SetTextAlign
        BOOL            m_bShow;
        DWORD           m_dwAlarmTick;       // Alarm for bubble erasing
};
#endif __BUBBLE_H

```

```

//
//  CBubbleNotifyObj:
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
//  ===== Nigel Thompson
//  This is a class of pure virtual functions with no data.
//  It is used by bubble objects to make notification callbacks.
//  A user of the CBubble class can derive an object from CBubbleNotifyObj
//  and pass a pointer to this derived class object to the bubble object for
//  notification calls.
//  Just like OLE's IClientSite interface really.
//

#ifndef __BUBBLENO_H
#define __BUBBLENO_H

class CBubble;

class AFX_EXT_CLASS CBubbleNotifyObj : public CObject
{
public:
    enum CHANGETYPE
    {
        ZORDER      = 0x0001,
        POSITION      = 0x0002,
        IMAGE        = 0x0004
    };

public:
    virtual void Change(CBubble* pBubble, CHANGETYPE change,
        CRect* pRect1=NULL, CRect* pRect2=NULL) =
        0;
};

#endif // __BUBBLENO_H

```

```

//
//  CBubbleList :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "dib.h"
#include "bubbleno.h"
#include "bubble.h"
#include "bblstno.h"
#include "bubblst.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CBubbleList
//   Ascending order
//   {}    <- draws last
//   {}
//   {}
//   ...
//   {}    <- draws first

IMPLEMENT_SERIAL(CBubbleList, CObList, 0 /* Schema number */)

CBubbleList::CBubbleList()
{
    // Give the bubble notification object
    // a pointer to the list object.
    m_NotifyObj.SetList(this);
}

CBubbleList::~CBubbleList()
{
}

////////////////////////////////////
// CBubbleList serialization

void CBubbleList::Serialize(CArchive& ar)
{
    // Let the base class create the set of objects.
    CObList::Serialize(ar);

    // If we've just loaded, initialize each bubble.
    if (ar.IsLoading())
    {
        for (POSITION pos = GetHeadPosition(); pos != NULL;)
        {
            CBubble* pBubble = GetNext(pos); // Increment position.
        }
    }
}

```

```

        pBubble->SetNotificationObject(&m_NotifyObj);
    }
}

////////////////////////////////////
// CBubbleList commands

// Add a bubble to the list, placing it according to its z-order value.
BOOL CBubbleList::Insert(CBubble* pNewBubble)
{
    // Set the notification object pointer in the bubble
    // to the bubble list's notification object.
    pNewBubble->SetNotificationObject(&m_NotifyObj);

    // Walk down the list until we either get to the end
    // or we find a bubble with the same or higher z-order
    // in which case we insert just before that one. Ascending order

    CBubble* pBubble;
    POSITION posThis;
    POSITION pos = GetHeadPosition();
    while (pos)
    {
        posThis = pos;
        pBubble = GetNext(pos); // Increment position.
        InsertBefore(posThis, pNewBubble);
        return TRUE;
    }
    // Nothing with the same or a higher z-order,
    // so add the bubble to the end.
    AddTail(pNewBubble);
    return TRUE;
}

// Remove a bubble from the list, but do not delete it
CBubble* CBubbleList::Remove(CBubble* pBubble)
{
    POSITION pos = Find(pBubble);
    if (pos == NULL)
        return NULL;
    RemoveAt(pos);
    return pBubble;
}

// Remove everything from the list deleting all the bubbles we remove
void CBubbleList::RemoveAll()
{
    // Walk down the list deleting objects as we go.
    // We need to do this here because the base class simply deletes the
    pointers.
    CBubble* pBubble;
    POSITION pos = GetHeadPosition();
    while (pos)
    {
        pBubble = GetNext(pos); // Increment position.
        if (pBubble)

```

```

        {
            ASSERT(pBubble->IsKindOf(RUNTIME_CLASS(CBubble)));
            delete pBubble;
        }
    }
    // Now call the base class to remove the pointers.
    CObList::RemoveAll();
}

// Move a bubble to its correct z-order position.
void CBubbleList::Reorder(CBubble* pBubble)
{
    // Remove the bubble from the list.
    if (!Remove(pBubble))
    {
        TRACE("Unable to find bubble");
        return; // Not there.
    }
    // Now insert it again in the right place.
    Insert(pBubble);
}

// Test for a mouse hit on any bubble in the list.
CBubble* CBubbleList::HitTest(const CPoint& point)
{
    // Walk the list top down.
    CBubble* pBubble;
    POSITION pos = GetHeadPosition();
    while (pos)
    {
        pBubble = GetNext(pos); // Increment position.
        if (pBubble->HitTest(point))
            return pBubble;
    }
    return NULL;
}

```

```

//
//  CBubbleList:
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#ifndef __BUBBLELST__
#define __BUBBLELST__

#include "BbLstNo.h"

class CBubble;

class AFX_EXT_CLASS CBubbleList : private CObList
{
    DECLARE_SERIAL(CBubbleList)
public:
    CBubbleList();
    ~CBubbleList();
    BOOL      Insert(CBubble* pBubble);
    void      Reorder(CBubble* pBubble);
    CBubble*  Remove(CBubble* pBubble);
    void      RemoveAll();
    CBubble*  GetNext(POSITION &pos) { return (CBubble*)
CObList::GetNext(pos); }
    CBubble*  GetPrev(POSITION &pos) { return (CBubble*)
CObList::GetPrev(pos); }
    POSITION  GetTailPosition() const { return
CObList::GetTailPosition(); }
    POSITION  GetHeadPosition() const { return
CObList::GetHeadPosition(); }
    CBubble*  HitTest(const CPoint& point);
    BOOL      IsEmpty() { return CObList::IsEmpty(); }
    virtual   void  Serialize(CArchive& ar);

public:
    CBubbleListNotifyObj m_NotifyObj;
};
#endif // __SPRITELIST__

```

```

//
//  CDIB :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
#include "stdafx.h"
#include "DIB.h"
#include "DIBPal.h"

#include <lzexpand.h>

#ifdef _VICTOR
#include <vicdefs.h>    // link Vic32ms.lib
#endif

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
// #if !defined(KSM_REPORT)
//     #define KSM_REPORT
// #endif
#endif

////////////////////////////////////
// CDIB

IMPLEMENT_SERIAL(CDIB, CObject, 0 /* Schema number */)

// Create a small DIB here so m_pBMI and m_pBits are always valid.
CDIB::CDIB()
{
    m_pBMI          = NULL;
    m_pBits          = NULL;
    m_bMyBits        = TRUE;
    m_bMapColorsDone = FALSE;
    m_strName.Empty();
    Create(16, 16);
}

CDIB::~CDIB()
{
    // Free the memory.
    if (m_pBMI)
        free(m_pBMI);
    if (m_bMyBits && m_pBits)
        free(m_pBits);
}

CDIB& CDIB::operator=(CDIB& rhs)
{
    if (this == &rhs)
        return *this;
    ASSERT(rhs.m_pBMI);
    ASSERT(rhs.m_pBits);

```

```

    BITMAPINFOHEADER* pBI = (BITMAPINFOHEADER*)rhs.m_pBMI;

    int nBMISize = sizeof(BITMAPINFOHEADER) + 256 * sizeof(RGBQUAD);
    m_pBMI = (BITMAPINFO*)malloc(nBMISize);
    ::CopyMemory(m_pBMI, rhs.m_pBMI, nBMISize);

    int iBitsSize = CountBitsSize(pBI->biWidth, pBI->biHeight);
    m_pBits = (BYTE*)malloc(iBitsSize);
    ::CopyMemory(m_pBits, rhs.m_pBits, iBitsSize);

    m_bMyBits          = rhs.m_bMyBits;
    m_bMapColorsDone   = rhs.m_bMapColorsDone;
    m_strName          = rhs.m_strName;
    return *this;
}

// CDIB serialization

// We don't support this yet.
void CDIB::Serialize(CArchive& ar)
{
    ar.Flush();
    CFile* fp = ar.GetFile();

    if (ar.IsStoring())
    {
        Save(fp);
    }
    else
    {
        Load(fp);
    }
}

// Private functions

static BOOL IsWinDIB(const BITMAPINFOHEADER *pBIH)
{
    ASSERT(pBIH);
    if (((BITMAPCOREHEADER*)pBIH)->bcSize == sizeof(BITMAPCOREHEADER))
        return FALSE;
    return TRUE;
}

static int NumDIBColorEntries(BITMAPINFO* pBmpInfo)
{
    BITMAPINFOHEADER* pBIH;
    BITMAPCOREHEADER* pBCH;
    int iColors, iBitCount;

    ASSERT(pBmpInfo);

    pBIH = &(pBmpInfo->bmiHeader);
    pBCH = (BITMAPCOREHEADER*)pBIH;

```



```

    // Start off by assuming the color table size from the bit-per-pixel
    field.
    iBitCount = IsWinDIB(pBIH) ? pBIH->biBitCount : pBCH->bcBitCount;

    switch (iBitCount)
    {
    case 1:          iColors = 2;          break;
    case 4:          iColors = 16;         break;
    case 8:          iColors = 256;        break;
    default:         iColors = 0;          break;
    }

    // If this is a Windows DIB, then the color table length is determined
    // by the biClrUsed field if the value in the field is nonzero.
    if (IsWinDIB(pBIH) && (pBIH->biClrUsed != 0))
        iColors = pBIH->biClrUsed;

    // BUGFIX 18 Oct 94 Nigelt
    // Make sure the value is reasonable since some products
    // will write out more than 256 colors for an 8 bpp DIB!!!
    int iMax = 0;
    switch (iBitCount)
    {
    case 1:          iMax = 2;          break;
    case 4:          iMax = 16;         break;
    case 8:          iMax = 256;        break;
    default:         iMax = 0;          break;
    }
    if (iMax)
    {
        if (iColors > iMax)
        {
            TRACE("Invalid color count\n");
            iColors = iMax;
        }
    }
    return iColors;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CDIB commands
#ifdef _DEBUG

#endif
// Create a new empty 8bpp DIB with a 256 entry color table.
BOOL CDIB::Create(const iWidth, const iHeight, LPCSTR szPalFileName)
{
    // Delete any existing stuff.
    if (m_pBMI)
        free(m_pBMI);
    if (m_bMyBits && m_pBits)
        free(m_pBits);

    // Allocate memory for the header.
    m_pBMI = (BITMAPINFO*)malloc(sizeof(BITMAPINFOHEADER) + 256 *
sizeof(RGBQUAD));
    if (!m_pBMI)

```

```

    {
        TRACE("Out of memory for DIB header\n");
        return FALSE;
    }

    // Allocate memory for the bits (DWORD aligned).
    int iBitsSize = CountBitsSize(iWidth, iHeight);
    m_pBits = (BYTE*)malloc(iBitsSize);
    if (!m_pBits)
    {
        TRACE("Out of memory for DIB bits\n");
        free(m_pBMI);
        m_pBMI = NULL;
        return FALSE;
    }
    m_bMyBits = TRUE;

    // Fill in the header info.
    BITMAPINFOHEADER* pBI = (BITMAPINFOHEADER*)m_pBMI;
    pBI->biSize = sizeof(BITMAPINFOHEADER);
    pBI->biWidth = iWidth;
    pBI->biHeight = iHeight;
    pBI->biPlanes = 1;
    pBI->biBitCount = 8;
    pBI->biCompression = BI_RGB;
    pBI->biSizeImage = 0;
    pBI->biXPelsPerMeter = 0;
    pBI->biYPelsPerMeter = 0;
    pBI->biClrUsed = 0;
    pBI->biClrImportant = 0;

    // Create an arbitrary color table (gray scale).
    if (szPalFileName)
    {
        LoadPalette(szPalFileName);
    }
    else
    {
        RGBQUAD* prgb = GetClrTabAddress();
        for (int i=0; i < 256; i++)
        {
            prgb->rgbBlue = prgb->rgbGreen = prgb->rgbRed = (BYTE)i;
            prgb->rgbReserved = 0;
            prgb++;
        }
    }
    // Set all the bits to a known state (black).
    ::ZeroMemory(m_pBits, iBitsSize);

    return TRUE;
}

int CDIB::GetBitsSize() const
{
    if (!m_pBMI)
        return 0;
    BITMAPINFOHEADER* pBI = (BITMAPINFOHEADER*)m_pBMI;

```

```

        return (((pBI->biWidth + 3) & ~3)*pBI->biHeight);
    }

void CDIB::ClearImage()
{
    if (m_pBits)
    {
        ::ZeroMemory(m_pBits, GetBitsSize());
    }
}

void CDIB::ClearRect(CRect& rcClear)
{
    if (!m_pBits)
        return;
    int w = rcClear.Width();
    int h = rcClear.Height();
    // Test for silly cases.
    if (w == 0 || h == 0)
        return;
    int xd = rcClear.left;
    int yd = rcClear.top;
    BYTE* pDest = (BYTE*)GetPixelAddress(xd, yd + h - 1);
    ASSERT(pDest);

    // Get the scan line widths of each DIB.
    int iDInc = StorageWidth(); // Source increment value
    while (h-- > 0) // Fill the lines
    {
        ::ZeroMemory(pDest, w);
        pDest += iDInc;
    }
}

// Create a CDIB structure from existing header and bits.
// The DIB won't delete the bits and makes a copy of the header.
BOOL CDIB::Create(BITMAPINFO* pBMI, BYTE* pBits)
{
    ASSERT(pBMI);
    ASSERT(pBits);
    if (m_pBMI)
        free(m_pBMI);
    m_pBMI = (BITMAPINFO*)malloc(sizeof(BITMAPINFOHEADER) +
256*sizeof(RGBQUAD));
    ASSERT(m_pBMI);
    // Note: This will probably fail for < 256 color headers.
    ::CopyMemory(m_pBMI, pBMI, sizeof(BITMAPINFOHEADER) +
NumDIBColorEntries(pBMI)*sizeof(RGBQUAD));

    if (m_bMyBits && m_pBits)
        free(m_pBits);
    m_pBits = pBits;
    m_bMyBits = FALSE; // We can't delete the bits.
    return TRUE;
}

// Load a palette from the file and replace existing one

```

```

BOOL CDIB::LoadPalette(LPCSTR szPalFileName)
{
    CDIBPal          pal;
    PALETTEENTRY      PE[256];
    BOOL bRes = pal.Load(szPalFileName, PE);
    SetPaletteEntries(0, 256, PE);

    /*
    CStdioFile f("Palette.txt", CFile::modeCreate | CFile::modeWrite |
CFile::typeText);
    CString str;
    for (int i=0; i < 256; i++)
    {
        str.Format("{d,%d,%d,%d},    // %d\n",
                    PE[i].peRed, PE[i].peGreen, PE[i].peBlue,
PE[i].peFlags, i);
        f.WriteString(str);
    }
    f.Close();
    */

    return bRes;
}

// Load a DIB from an open file.
// Soomin Kim made the following BMP variations
// BMZ: LZ compressed BMP file
// BM: Excluding palette section from BMP
// BMC: Compressed BM file
BOOL CDIB::Load(CFile* const fp, LPCSTR szPalFileName, const BOOL bIsLZ,
const DWORD dwFileStart)
{
    BOOL bIsPM = FALSE;
    BOOL bIsBMC = FALSE;
    BITMAPINFO* pBmpInfo = NULL;
    BYTE* pBits = NULL;
    UINT hLZFile;

    // Read the file header to get the file size and to find where the bits
start in the file.
    BITMAPFILEHEADER BmpFileHdr;
    int iBytes;
    if (bIsLZ) // (BmpFileHdr.bfType == 0x5a53)    // 'SZ'
    {
        hLZFile = ::LZInit(fp->m_hFile);
        // long lFileSize = ::LZSeek(hFile, 0L, 2); // points to the end of
file
        // TRACE("Compressed BMP: %ld bytes.\n", lFileSize);
        ::LZSeek(hLZFile, dwFileStart, 0); // seek to beginning of file
        iBytes = ::LZRead(hLZFile, (char*)&BmpFileHdr,
sizeof(BmpFileHdr));
    }
    else
    {
        fp->Seek(dwFileStart, CFile::begin);    // Rewind
        iBytes = fp->Read(&BmpFileHdr, sizeof(BmpFileHdr));
    }
    if (iBytes != sizeof(BmpFileHdr))
    {

```

```

        TRACE("Failed to read file header\n");
        goto $abort;
    }

    if (BmpFileHdr.bfType == 0x4342)    // 'BC'
    {
        bIsBMC = TRUE;    // No palette
    }
    else if (BmpFileHdr.bfType != 0x4d42)    // 'BM'
    {
        TRACE("Not a bitmap file\n");
        goto $abort;
    }

    // Make a wild guess that the file is in Windows DIB format and read
    the BITMAPINFOHEADER.
    // If the file turns out to be a PM DIB file we'll convert it later.
    BITMAPINFOHEADER BmpInfoHdr;
    if (bIsLZ)
        iBytes = ::LZRead(hLZFile, (char*)&BmpInfoHdr,
sizeof(BmpInfoHdr));
    else
        iBytes = fp->Read(&BmpInfoHdr, sizeof(BmpInfoHdr));
    if (iBytes != sizeof(BmpInfoHdr))
    {
        TRACE("Failed to read BITMAPINFOHEADER\n");
        goto $abort;
    }

    // Check that we got a real Windows DIB file.
    if (BmpInfoHdr.biSize != sizeof(BITMAPINFOHEADER))
    {
        if (BmpInfoHdr.biSize != sizeof(BITMAPCOREHEADER))
        {
            TRACE(" File is not Windows or PM DIB format\n");
            goto $abort;
        }

        // Set a flag to convert PM file to Win format later.
        bIsPM = TRUE;

        // Back up the file pointer and read the BITMAPCOREHEADER
        // and create the BITMAPINFOHEADER from it.
        if (bIsLZ)
            ::LZSeek(hLZFile, dwFileStart + sizeof(BITMAPFILEHEADER),
0);
        else
            fp->Seek(dwFileStart + sizeof(BITMAPFILEHEADER),
CFile::begin);
        BITMAPCOREHEADER BmpCoreHdr;
        if (bIsLZ)
            iBytes = ::LZRead(hLZFile, (char*)&BmpCoreHdr,
sizeof(BmpCoreHdr));
        else
            iBytes = fp->Read(&BmpCoreHdr, sizeof(BmpCoreHdr));
        if (iBytes != sizeof(BmpCoreHdr))
        {

```

```

        TRACE("Failed to read BITMAPCOREHEADER\n");
        goto $abort;
    }

    BmpInfoHdr.biSize          = sizeof(BITMAPINFOHEADER);
    BmpInfoHdr.biWidth         = (int)BmpCoreHdr.bcWidth;
    BmpInfoHdr.biHeight        = (int)BmpCoreHdr.bcHeight;
    BmpInfoHdr.biPlanes        = BmpCoreHdr.bcPlanes;
    BmpInfoHdr.biBitCount       = BmpCoreHdr.bcBitCount;
    BmpInfoHdr.biCompression   = BI_RGB;
    BmpInfoHdr.biSizeImage      = 0;
    BmpInfoHdr.biXPelsPerMeter  = 0;
    BmpInfoHdr.biYPelsPerMeter  = 0;
    BmpInfoHdr.biClrUsed        = 0;
    BmpInfoHdr.biClrImportant   = 0;
}

// Work out how much memory we need for the BITMAPINFO structure, color
table
// and then for the bits. Allocate the memory blocks.
// Copy the BmpInfoHdr we have so far, and then read in the color table
from the file.
    int iColors;
    int iColorTableSize;
    iColors          =
NumDIBColorEntries((LPBITMAPINFO)&BmpInfoHdr);
    iColorTableSize  = iColors * sizeof(RGBQUAD);
    // Always allocate enough room for 256 entries.
    int iBISize;
    int iBitsSize;
    iBISize          = sizeof(BITMAPINFOHEADER) + 256 * sizeof(RGBQUAD);
    iBitsSize        = BmpFileHdr.bfSize - BmpFileHdr.bfOffBits;

// TRACE("CDIB::Load(\"%s\") size=%ld\n", m_strName, BmpFileHdr.bfSize);

// Allocate the memory for the header.
// =====
pBmpInfo = (LPBITMAPINFO)malloc(iBISize);
if (!pBmpInfo)
{
    TRACE("Out of memory for DIB header\n");
    goto $abort;
}

// Copy the header we already have.
::CopyMemory(pBmpInfo, &BmpInfoHdr, sizeof(BITMAPINFOHEADER));

// Now read the color table from the file.
if (bIsBMC)
{
    ::ZeroMemory(((LPBYTE)pBmpInfo) + sizeof(BITMAPINFOHEADER),
iColorTableSize);
}
else
{
    if (!bIsPM)
    {

```

```

        // Read the color table from the file.
        if (bIsLZ)
            iBytes = ::LZRead(hLZFile, (char*)((LPBYTE)pBmpInfo
+ sizeof(BITMAPINFOHEADER)), iColorTableSize);
        else
            iBytes = fp->Read((LPBYTE)pBmpInfo +
sizeof(BITMAPINFOHEADER), iColorTableSize);
        if (iBytes != iColorTableSize)
        {
            TRACE("Failed to read color table\n");
            goto $abort;
        }
    }
    else
    {
        // Read each PM color table entry in turn and convert it to
Win DIB format as we go.
        LPRGBQUAD lpRGB = (LPRGBQUAD)((LPBYTE)pBmpInfo +
sizeof(BITMAPINFOHEADER));
        RGBTRIPLE rgbt;
        for (int i=0; i < iColors; i++)
        {
            if (bIsLZ)
                iBytes = ::LZRead(hLZFile, (char*)&rgbt,
sizeof(RGBTRIPLE));
            else
                iBytes = fp->Read(&rgbt, sizeof(RGBTRIPLE));
            if (iBytes != sizeof(RGBTRIPLE))
            {
                TRACE("Failed to read RGBTRIPLE\n");
                goto $abort;
            }
            lpRGB->rgbBlue = rgbt.rgbtBlue;
            lpRGB->rgbGreen = rgbt.rgbtGreen;
            lpRGB->rgbRed = rgbt.rgbtRed;
            lpRGB->rgbReserved = 0;
            lpRGB++;
        }
    }
}

// Allocate the memory for the bits and read the bits from the file.
// =====
pBits = (BYTE*)malloc(iBitsSize);
if (!pBits)
{
    TRACE("Out of memory for DIB bits\n");
    goto $abort;
}

// Seek to the bits in the file.
if (bIsLZ)
    ::LZSeek(hLZFile, dwFileStart + BmpFileHdr.bfOffBits, 0);
else
    fp->Seek(dwFileStart + BmpFileHdr.bfOffBits, CFile::begin);

// Read the bits.

```

```

        if (bIsLZ)
            iBytes = ::LZRead(hLZFile, (char*)pBits, iBitsSize);
        else
            iBytes = fp->Read(pBits, iBitsSize);
        if (iBytes != iBitsSize)
        {
            TRACE("Failed to read bits\n");
            goto $abort;
        }

        // Everything went OK.
        if (bIsLZ)
            ::LZClose(hLZFile);
        if (m_pBMI)
            free(m_pBMI);
        m_pBMI = pBmpInfo;
        if (m_bMyBits && m_pBits)
            free(m_pBits);
        m_pBits = pBits;
        m_bMyBits = TRUE;

        if (szPalFileName) // Overload the palette entries with that of
the specified file
        {
            LoadPalette(szPalFileName);
        }
        return TRUE;

$abort: // Something went wrong.
        if (bIsLZ)
            ::LZClose(hLZFile);
        if (pBmpInfo)
            free(pBmpInfo);
        if (pBits)
            free(pBits);
        return FALSE;
    }

    // Load a DIB from a disk file.
    // If no file name is given, show an Open File dialog to get one.
    // m_strName == "DATASRC0:filename.ext"
    BOOL CDIB::Load(LPCSTR szFileName, LPCSTR szPalFileName)
    {
        if ((szFileName == NULL) || (strlen(szFileName) == 0))
        {
            // Show an Open File dialog to get the name.
            CFileDialog dlg(TRUE, // Open
                            NULL, // No default extension
                            NULL, // No initial file name
                            OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,

#ifdef _VICTOR
                            "Bitmap
files|*.DIB;*.BMP;*.BM;*.gif;*.jpg;*.tif|All files (*.*)|*.*||");
#else
                            "Bitmap
files(*.DIB;*.BM*)|*.DIB;*.BM*|All files (*.*)|*.*||");
#endif // _VICTOR
            if (dlg.DoModal() == IDOK)
                m_strName = dlg.GetPathName();
            else

```



```

        return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        m_strName = szFileName;
    }

    // Parse Data Source
    // tile0000|tcity000.bm
    int iDS = m_strName.Find('|');
    if (iDS > 0)
    {
        int nLen = m_strName.GetLength();
        CString strDS(m_strName.Left(iDS));
        strDS += ".uds";
        CString strBM(m_strName.Right(nLen - (iDS + 1)));
        return LoadDataSource(strDS, strBM, szPalFileName);
    }

    BOOL bResult;
#ifdef _VICTOR
    int nFT = GetBitmapFileType(m_strName);
    if ((nFT == FILE_BMP) || (nFT == FILE_BM) ||
        (nFT == FILE_BMZ) || (nFT == FILE_BMC))
    {
#endif // _VICTOR
        // Try to open the file for read access.
        CFile file;
        if (!file.Open(m_strName, CFile::modeRead |
            CFile::shareDenyWrite))
        {
            TRACE1("File Open Failure: %s\n", m_strName);
            return FALSE;
        }

        WORD wFileType;
        int iBytes = file.Read(&wFileType, sizeof(wFileType));
        if (iBytes != sizeof(wFileType))
        {
            TRACE("Failed to read file\n");
            file.Close();
            return FALSE;
        }

        BOOL bIsLZ = (wFileType == 0x5a53); // 'SZ'

        bResult = Load(&file, szPalFileName, bIsLZ, 0L);
        file.Close();
#ifdef _VICTOR
    }
    else
    {
        imgdes orgImg;    // original image
        imgdes cnvImg;    // converted image (24bit color -> 8bit color)
        switch (nFT)
        {

```

```

        case FILE_GIF:
            LoadGIF(m_strName, &orgImg);
            break;
        case FILE_TIF:
            LoadTIF(m_strName, &orgImg);
            break;
        case FILE_JPG:
            LoadJPG(m_strName, &orgImg);
            break;
    }
    if (orgImg.bmh->biBitCount != 8)
    {
        if (CreateImage8(&orgImg, &cnvImg) != NO_ERROR)
        {
            AfxMessageBox("Image color dithering failure.");
            return FALSE;
        }
        freeimage(&orgImg);
        bResult = LoadFromImage((LPVOID)cnvImg.bmh);
        freeimage(&cnvImg);
    }
    else
    {
        bResult = LoadFromImage((LPVOID)orgImg.bmh);
        freeimage(&orgImg);
    }
}
#endif // _VICTOR

    return bResult;
}

BOOL CDIB::LoadDataSource(LPCSTR szSrcFileName,
                          LPCSTR szFileName,
                          LPCSTR szPalFileName)
{
    BOOL bResult;
    // Try to open the file for read access.
    CFile file;
    if (!file.Open(szSrcFileName, CFile::modeRead | CFile::shareDenyWrite))
    {
        TRACE1("File Open Failure: %s\n", szSrcFileName);
        return FALSE;
    }

    BOOL bIsLZ = FALSE;
    UINT hLZFile;
    DATASOURCEFILEHEADER dsFH;

    int iBytes = file.Read(&dsFH, sizeof(dsFH));
    if (iBytes != sizeof(dsFH))
    {
        TRACE("Failed to read file header\n");
        file.Close();
        return FALSE;
    }
}

```

```

        if (dsFH.wType == 0x5a53)        // 'SZ'        - test for compression
        {
            bIsLZ = TRUE;
            hLZFile = ::LZInit(file.m_hFile);
            // long lFileSize = ::LZSeek(hFile, 0L, 2); // points to the end of
file
            // TRACE("Compressed BMP: %ld bytes.\n", lFileSize);
            ::LZRead(hLZFile, (char*)&dsFH, sizeof(dsFH)); // read again
        }
        else if (dsFH.wType != 0x5344)    // 'DS'
        {
            TRACE("Wrong filetype.\n");
            file.Close();
            return FALSE;
        }

// TRACE("Data Source F. Size: %ld\n", dsFH.dwFileSize);
// TRACE("Data Source Entries: %d\n", dsFH.wNumEntries);
DATASOURCEENTRY* aDSE = new DATASOURCEENTRY[dsFH.wNumEntries];

        if (bIsLZ)
            ::LZRead(hLZFile, (char*)aDSE, sizeof(DATASOURCEENTRY) *
dsFH.wNumEntries);
        else
            file.Read(aDSE, sizeof(DATASOURCEENTRY) * dsFH.wNumEntries);

        for (WORD i=0; i < dsFH.wNumEntries; i++)
        {
            if (lstrcmpi(aDSE[i].id, szFileName) == 0)    // matching!
            {
                // if (bIsLZ)
                //         ::LZClose(hLZFile);
                bResult = Load(&file, szPalFileName, bIsLZ,
aDSE[i].dwOffset);
                delete [] aDSE;
                file.Close();
                return bResult;
            }
        }
        TRACE("CDIB: %s not found in data source %s", szFileName,
szSrcFileName);
        delete [] aDSE;
        file.Close();
        return FALSE;
    }

// Load a DIB from a resource id.
BOOL CDIB::Load(const WORD wResid)
{
    ASSERT(wResid);
    HINSTANCE hInst = AfxGetResourceHandle();
    HRSRC hrsrc = ::FindResource(hInst, MAKEINTRESOURCE(wResid), "DIB");
    if (!hrsrc)
    {
        TRACE1("DIB resource not found(HRSRC:%x)", hrsrc);
        return FALSE;
    }
}

```

```

HGLOBAL hg = ::LoadResource(hInst, hrsrc);
if (!hg)
{
    TRACE1("Failed to load DIB resource(HGLOBAL:%x)", hg);
    return FALSE;
}

m_strName.Format("RES:%d", wResid);

BYTE* pRes = (BYTE*)::LockResource(hg);
ASSERT(pRes);
int iSize = ::SizeofResource(hInst, hrsrc);

// Mark the resource pages as read/write so the mmioOpen won't fail
DWORD dwOldProt;
BOOL b = ::VirtualProtect(pRes, iSize, PAGE_READWRITE, &dwOldProt);
ASSERT(b);

// Now create the CDIB object. We will create a new header from the
data
// in the resource image and copy the bits from the resource to a new
block of memory.
// We can't use the resource image as-is because we might want to map
the DIB colors
// and the resource memory is write protected in Win32.
BITMAPFILEHEADER* pFileHdr = (BITMAPFILEHEADER*)pRes;
ASSERT(pFileHdr->bfType == 0x4D42); // BM file
BITMAPINFOHEADER* pInfoHdr = (BITMAPINFOHEADER*)(pRes +
sizeof(BITMAPFILEHEADER));
ASSERT(pInfoHdr->biSize == sizeof(BITMAPINFOHEADER)); // must be a Win
DIB
BYTE* pBits = pRes + pFileHdr->bfOffBits;
BOOL bResult = Create((BITMAPINFO*)pInfoHdr, pBits);
return bResult;
// Note: not required to unlock or free the resource in Win32
}

// Draw the DIB to a given DC.
void CDIB::Draw(CDC* pDC, const x, const y)
{
    ::StretchDIBits(pDC->GetSafeHdc(),
        x, y, DibWidth(), DibHeight(), // Destination x, y,
width, height
        0, 0, DibWidth(), DibHeight(), // Source x, y, width,
height
        GetBitsAddress(), // Pointer to bits
        GetBitmapInfoAddress(), // BITMAPINFO
        DIB_RGB_COLORS, // Options
        SRCCOPY); // Raster
operation code (ROP)
}

// Draw a portion of source DIB to the destination
void CDIB::Draw(CDC* pDC, const xd, const yd, const w, int h, const xs, const
ys)
{
    ::StretchDIBits(pDC->GetSafeHdc(),

```

```

        xd, yd, w, h, // Destination x,
y, width, height    xs, DibHeight()-(ys+h), w, h, // Source x, y, width, height
                    GetBitsAddress(), // Pointer to bits
                    GetBitmapInfoAddress(), // BITMAPINFO
                    DIB_RGB_COLORS, // Options
                    SRCCOPY); // Raster
operation code (ROP)
}

// Tile
void CDIB::Tile(CDC* pDC, const x0, const y0, CRect& rcClient)
{
    for (int y=y0; y <= rcClient.bottom; y += DibHeight())
        for (int x=x0; x <= rcClient.right; x += DibWidth())
            Draw(pDC, x, y);
}

// Get the number of color table entries.
int CDIB::GetNumClrEntries() const
{
    return NumDIBColorEntries(m_pBMI);
}

// map the colors in this DIB to the identity palette specified
// NOTE: This assumes all CDIB objects have 256 color table entries.
BOOL CDIB::MapColorsToPalette(const CPalette* pPal)
{
    if (m_bMapColorsDone)
        return TRUE;
    if (!pPal)
    {
        TRACE("No palette to map to\n");
        return FALSE;
    }
    ASSERT(m_pBMI);
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    ASSERT(m_pBits);
    LPRGBQUAD pctThis = GetClrTabAddress(); // We can redirect the
palette address // with
SetClrTabAddress(...) function.
    ASSERT(pctThis);
    // Build an index translation table to map this DIBs colors to those of
the reference DIB.
    BYTE imap[256];
#ifdef _DEBUG
    int iChanged = 0; // For debugging only
#endif
    for (int i=0; i < 256; i++)
    {
        imap[i] = (BYTE)pPal->GetNearestPaletteIndex(
            RGB(pctThis->rgbRed,
                pctThis->rgbGreen,
                pctThis->rgbBlue));
        pctThis++;
    }
#ifdef _DEBUG

```

```

        if (imap[i] != i)
            iChanged++; // For debugging
    }
    // TRACE("CDIB::MapColorsToPalette changed %d colors.\n", iChanged);
    #else
    }
    #endif
    // Now map the DIB bits.
    BYTE* pBits = (BYTE*)GetBitsAddress();
    int iSize = StorageWidth() * DibHeight();
    while (iSize--)
    {
        *pBits = imap[*pBits];
        pBits++;
    }
    // Now reset the DIB color table so that its RGB values match those in
    the palette.
    PALETTEENTRY pe[256];
    pPal->GetPaletteEntries(0, 256, pe);
    pctThis = GetClrTabAddress();
    for (i=0; i < 256; i++)
    {
        pctThis->rgbRed = pe[i].peRed;
        pctThis->rgbGreen = pe[i].peGreen;
        pctThis->rgbBlue = pe[i].peBlue;
        pctThis++;
    }
    // Now say all the colors are in use
    m_pBMI->bmiHeader.biClrUsed = 256;
    m_bMapColorsDone = TRUE;
    return TRUE;
}

// Get a pointer to a pixel.
// NOTE: DIB scan lines are DWORD aligned.
// The scan line storage width may be wider than the scan line image width
// so calc the storage width by rounding the image width to the next highest
DWORD value.
void* CDIB::GetPixelAddress(const x, const y) const
{
    // Note: This version deals only with 8 bpp DIBs.
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    // Make sure it's in range and if it isn't return zero.
    if ((x >= DibWidth()) || (y >= DibHeight()))
    {
        TRACE("Attempt to get out of range pixel address\n");
        return NULL;
    }

    // Calculate the scan line storage width.
    int iWidth = StorageWidth();
    return m_pBits + (DibHeight()-y-1) * iWidth + x;
}

// Get the bounding rectangle.
void CDIB::GetRect(CRect& rect) const
{

```

```

    rect.left = rect.top = 0;
    rect.right = DibWidth();
    rect.bottom = DibHeight();
}

// Copy a rectangle of the DIB to another DIB.
// Note: We only support 8bpp DIBs here.
void CDIB::CopyBits(CDIB* pdibDest, const xd, const yd,
                   const w, int h, const xs, const ys, const COLORREF
clrTrans,
                   const WORD wImOp)
{
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    ASSERT(pdibDest);
    WORD opacity = wImOp & OPACITY_MASK;
    // Test for silly cases.
    if (!w || !h || (opacity == OPACITY_0))
        return;

    BOOL bMirror          = wImOp & IMAGE_FLIP;
    BOOL bVT              = wImOp & IMAGE_VERTICAL;    // Vertical
Transformation

    // Get pointers to the start points in the source and destination DIBs.
    // Note that the start points will be the bottom-left corner of the
DIBs
    // because the scan lines are reversed in memory.
    BYTE* pSrc = (BYTE*)GetPixelAddress(xs, ys+h-1);
    ASSERT(pSrc);
    BYTE* pDest = (BYTE*)pdibDest->GetPixelAddress(xd, (bVT) ? yd : (yd+h-
1));
    ASSERT(pDest);

    // Get the scan line widths of each DIB.
    int iScanS = StorageWidth();
    int iScanD = pdibDest->StorageWidth();
    int iSInc = bMirror ? iScanS : (iScanS - w); // Source increment value
    int iDInc = bVT ? (-iScanD - w) : (iScanD - w); // Destination
increment value
    int iCount;
    BYTE pixel;

    if (wImOp & NO_COLORKEY)    // && opacity == OPACITY_100)
    {
        if (bMirror)
        {
            while (h--) // Copy the lines
            {
                iCount      = w;    // Number of pixels to scan.
                pSrc += w;
                while (iCount--)
                {
                    pixel      = *(--pSrc);
                    *pDest++    = pixel;
                }
                pSrc += iSInc;
                pDest += iDInc;
            }
        }
    }
}

```

```

    }
}
else // !Mirror
{
    while (h--) // Copy the lines
    {
        ::CopyMemory(pDest, pSrc, w);
        pSrc += iScanS;
        if (bVT)
            pDest -= iScanD;
        else
            pDest += iScanD;
    }
}
}
else // Use Colorkey
{
    // Copy lines with transparency.
    // Note: We accept only a PALETTEINDEX description for the color
definition.
    ASSERT((clrTrans & 0xFF000000) == 0x01000000);
    BYTE bTransClr = LOBYTE(LOWORD(clrTrans));

    if (opacity == OPACITY_100)
    {
        if (bMirror)
        {
            while (h--)
            {
                iCount = w;    // Number of pixels to scan.
                pSrc += w;
                while (iCount--)
                {
                    pixel = *(--pSrc);          // *pSrc--
                    // Copy pixel only if it isn't
transparent.

                    if (pixel != bTransClr) *pDest++ = pixel;
                    else
                        pDest++;
                }
                // Move on to the next line.
                pSrc += iSInc;
                pDest += iDInc;
            }
        }
        else // !bMirror
        {
            while (h--)
            {
                iCount = w;    // Number of pixels to scan.
                while (iCount--)
                {
                    pixel = *pSrc++;
                    // Copy pixel only if it isn't
transparent.

                    if (pixel != bTransClr) *pDest++ = pixel;
                    else
                        pDest++;
                }
            }
        }
    }
}

```



```

    }
    // Move on to the next line.
    pSrc += iSInc;
    pDest += iDInc;
}
}
}
else if (opacity == OPACITY_75)
{
    int r;
    int d=4;    // 3/4

    if (bMirror)
    {
        while (h--)
        {
            iCount = w;    // Number of pixels to scan.
            r = h % d;    // 0,1,2,3,0,... =>

            if (r==1)
                r = 2;
            else if (r==2)
                r = 1;
            pSrc += w;
            while (iCount--)
            {
                pixel = *--pSrc;
                // Copy pixel only if it isn't
                if ((iCount % d != r) && (pixel !=
                    *pDest++ = pixel;
                else
                    pDest++;
            }
            // Move on to the next line.
            pSrc += iSInc;
            pDest += iDInc;
        }
    }
    else // !bMirror
    {
        while (h--)
        {
            iCount = w;    // Number of pixels to scan.
            r = h % d;
            if (r==1)
                r = 2;
            else if (r==2)
                r = 1;
            while (iCount--)
            {
                pixel = *pSrc++;
                // Copy pixel only if it isn't
                if ((iCount % d != r) && (pixel !=
                    transparent.
                    bTransClr))

```

```

        *pDest++ = pixel;
    else
        pDest++;
    }
    // Move on to the next line.
    pSrc += iSInc;
    pDest += iDInc;
}
}
else //      !(OPACITY_100, OPACITY_75)
{
    int r;
    int d;

    switch (opacity)
    {
    case OPACITY_12: d = 4;          break;          //      1/64 paint
    case OPACITY_25:                               //      1/4
    case OPACITY_50: d = 2;          break;          //      1/2
    default:        d = 2;          break;
    }

    if (bMirror)
    {
        while (h--)
        {
            iCount = w; // Number of pixels to scan.
            r = h % d; // Since adjacent lines should be
different
                        pSrc += w;
                        while (iCount--)
                        {
                            if ((opacity != OPACITY_50) && (r != 0))
                            {
                                pSrc -= (iCount+1);
                                pDest += (iCount+1);
                                break; // skip this scan
line
                                }
                                pixel = *--pSrc;
                                // Copy pixel only if it isn't
transparent.
                                if ((iCount % d == r) && (pixel !=
bTransClr))
                                    *pDest++ = pixel;
                                else
                                    pDest++;
                            }
                            // Move on to the next line.
                            pSrc += iSInc;
                            pDest += iDInc;
                        }
                    }
                }
            else // !Mirror
            {
                while (h--)

```

```

    {
        iCount = w;    // Number of pixels to scan.
        r = h % d;
        while (iCount--)
        {
            if ((opacity != OPACITY_50) && (r != 0))
            {
                pSrc += (iCount+1);    // since 1
                pDest += (iCount+1);
                break;    // skip this scan
            }
            pixel = *pSrc++;
            // Copy pixel only if it isn't
            if ((iCount % d == r) && (pixel !=
                *pDest++ = pixel;
            else
                pDest++;
        }
        // Move on to the next line.
        pSrc += iSInc;
        pDest += iDInc;
    }
}

#ifdef KSM_REPORT
    static int D_CopyBitsCount = 0;
    TRACE3("CDIB::CopyBits(%d,%d)=%d\n", w, h, ++D_CopyBitsCount);
#endif
}

// Save a DIB to a disk file.
// This is somewhat simplistic because we only deal with 256 color DIBs
// and we always write a 256 color table.
BOOL CDIB::Save(CFile* const fp, const BOOL bPalette)
{
    BITMAPFILEHEADER bfh;

    // Construct the file header.
    bfh.bfType = (bPalette) ? 0x4D42 : 0x4342; // 'BM' or 'BC'
    bfh.bfSize =
        sizeof(BITMAPFILEHEADER) +
        sizeof(BITMAPINFOHEADER) +
        StorageWidth() * DIBHeight();
    if (bPalette)
        bfh.bfSize += 256 * sizeof(RGBQUAD);
    bfh.bfReserved1 = 0;
    bfh.bfReserved2 = 0;
    bfh.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
    if (bPalette)
        bfh.bfOffBits += 256 * sizeof(RGBQUAD);

    // Write the file header.

```

```

    int iSize = sizeof(bfh);
    TRY
    {
        fp->Write(&bfh, iSize);
    }
    CATCH(CFileException, e)
    {
        TRACE("Failed to write file header");
        return FALSE;
    } END_CATCH

    // Write the BITMAPINFO structure.
    // Note: we assume that there are always 256 colors in the color table.
    ASSERT(m_pBMI);
    iSize = sizeof(BITMAPINFOHEADER);
    if (bPalette)
        iSize += 256 * sizeof(RGBQUAD);
    TRY
    {
        fp->Write(m_pBMI, iSize);
    }
    CATCH(CFileException, e)
    {
        TRACE("Failed to write BITMAPINFO");
        return FALSE;
    } END_CATCH

    // Write the bits.
    iSize = StorageWidth() * DibHeight();
    TRY
    {
        fp->Write(m_pBits, iSize);
    }
    CATCH(CFileException, e)
    {
        TRACE("Failed to write bits");
        return FALSE;
    } END_CATCH

    return TRUE;
}

// Save a DIB to a disk file. If no file name is given, show a File Save
// dialog to get one.
BOOL CDIB::Save(LPCSTR szFileName)
{
    CString strFile;

    if ((szFileName == NULL) || (strlen(szFileName) == 0))
    {
        // Show a File Save dialog to get the name.
        CFileDialog dlg(FALSE, // Save
                        NULL, // No default extension
                        m_strName, // No initial file name
                        OFN_OVERWRITEPROMPT | OFN_HIDEREADONLY,
                        "Image files
(*.DIB;*.BM*)|*.DIB;*.BM*|All files (*.*)|*.*||");
    }

```

```

        if (dlg.DoModal() == IDOK)
        {
            strFile = dlg.GetPathName();
        }
        else
        {
            return FALSE;
        }
    }
    else
    {
        // Copy the supplied file path.
        strFile = szFileName;
    }

    // Try to open the file for write access.
    CFile file;
    if (!file.Open(strFile, CFile::modeReadWrite | CFile::modeCreate |
CFile::shareExclusive))
    {
        AfxMessageBox("Failed to open file");
        return FALSE;
    }

    BOOL bResult = Save(&file, (GetFileType(strFile) != FILE_BM));
    file.Close();
    if (!bResult)
        AfxMessageBox("Failed to save file");
    return bResult;
}

CDIB::BITMAP_FILE_TYPE CDIB::GetFileType(LPCSTR szFile)
{
    char* p = strrchr(szFile, '.');    // .extension
    if (p)
    {
        p++;
        if (lstrcmpi(p, "BM") == 0)    // BMP, BMZ, BMC, ...
        {
            switch (toupper(p[2]))
            {
                case NULL: return FILE_BM;
                case 'Z': return FILE_BMZ;
                case 'C': return FILE_BMC;
                default: return FILE_BMP; // 'p'
            }
        }
        if (lstrcmpi(p, "GIF") == 0)
            return FILE_GIF;
        if (lstrcmpi(p, "TIF") == 0)
            return FILE_TIF;
        if (lstrcmpi(p, "JPG") == 0)
            return FILE_JPG;
    }
    char szMsg[256];
    wsprintf(szMsg, "Unrecognized bitmap file type - \"%s\"", szFile);
    AfxMessageBox(szMsg);
}

```

```

        return FILE_NONE;
    }

#ifdef _VICTOR
////////////////////////////////////
////////////////////////////////////
// Victor

// Load a DIB from an open file.
BOOL CDIB::LoadFromImage(LPVOID pImg)
{
    BYTE* pSrc = (BYTE*)pImg;
    // Work out how much memory we need for the BITMAPINFO structure, color
table
    // and then for the bits. Allocate the memory blocks.
    // Always allocate enough room for 256 entries.
    // Allocate the memory for the header.
    BITMAPINFO* pBmpInfo = (LPBITMAPINFO)malloc(sizeof(BITMAPINFOHEADER) +
256 * sizeof(RGBQUAD));
    if (!pBmpInfo)
    {
        TRACE("Out of memory for DIB header");
        return FALSE;
    }
    // Copy the header
    ::CopyMemory(pBmpInfo, pSrc, sizeof(BITMAPINFOHEADER));
    pSrc += sizeof(BITMAPINFOHEADER);

    int iBitsSize      = pBmpInfo->bmiHeader.biSizeImage;
    // Copy the BmpInfoHdr we have so far, and then read in the color table
from the file.
    int iColorTableSize      = NumDIBColorEntries(pBmpInfo) *
sizeof(RGBQUAD);

    // Read the color table from the file.
    ::CopyMemory(((LPBYTE)pBmpInfo) + sizeof(BITMAPINFOHEADER), pSrc,
iColorTableSize);
    pSrc += iColorTableSize;

    // Allocate the memory for the bits and read the bits from the file.
    BYTE* pBits = (BYTE*)malloc(iBitsSize);
    if (!pBits)
    {
        TRACE("Out of memory for DIB bits");
        if (pBmpInfo)
            free(pBmpInfo);
        return FALSE;
    }

    // Read the bits.
    ::CopyMemory(pBits, pSrc, iBitsSize);

    // Everything went OK.
    if (m_pBMI)
        free(m_pBMI);
    m_pBMI = pBmpInfo;
    if (m_bMyBits && m_pBits)

```

```

        free(m_pBits);
        m_pBits      = pBits;
        m_bMyBits    = TRUE;
        return TRUE;
    }

int CDIB::LoadGIF(LPCSTR fname, imgdes* image)
{
    // Get info on the file we're to load
    GifData gdat;
    CString str;
    int rcode = gifinfo((char*)fname, &gdat);
    if (rcode != NO_ERROR) // Fill structure
    {
        str.Format("Error in reading %s", fname);
        AfxMessageBox(str);
        return rcode;
    } // Allocate space for an 8-bit image
    rcode = allocimage(image, gdat.width, gdat.length, gdat.vbitcount);
    if (rcode != NO_ERROR)
    {
        str.Format("Not enough memory to load %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Load image
    rcode = loadgif(fname, image);
    if (rcode != NO_ERROR)
    {
        freeimage(image); // Free image on error
        str.Format("Could not load %s", fname);
        AfxMessageBox(str);
    }
    return rcode;
}

int CDIB::LoadTIF(LPCSTR fname, imgdes* image)
{
    // Get info on the file we're to load
    TiffData tdat;
    CString str;
    int rcode = tiffinfo((char*)fname, &tdat);
    if (rcode != NO_ERROR) // Fill structure
    {
        str.Format("Error in reading %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Allocate space for an image
    rcode = allocimage(image, tdat.width, tdat.length, tdat.vbitcount);
    if (rcode != NO_ERROR)
    {
        str.Format("Not enough memory to load %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    rcode = loadtif(fname, image);
}

```

```

    if (rcode != NO_ERROR) // Free image on error
    {
        freeimage(image);
        str.Format("Could not load %s", fname);
        AfxMessageBox(str);
    }
    return rcode;
}

int CDIB::LoadJPG(LPCSTR fname, imgdes* image)
{
    // Get info on the file we're to load
    JpegData jdat;
    CString str;
    int rcode = jpeginfo(fname, &jdat);
    if (rcode != NO_ERROR) // Fill structure
    {
        str.Format("Error in reading %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Allocate space for an image
    rcode = allocimage(image, (int)jdat.width, (int)jdat.length,
jdat.vbitcount);
    if (rcode != NO_ERROR)
    {
        str.Format("Not enough memory to load %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Load image
    rcode = loadjpg((char*)fname, image);
    if (rcode != NO_ERROR)
    {
        freeimage(image); // Free image on error
        str.Format("Could not load %s", fname);
        AfxMessageBox(str);
    }
    return rcode;
}

// Source image will remain intact.
// Destination image should be a pointer to an imgdes object.
// You should call freeimage(desimg) on success
int CDIB::CreateImage8(imgdes* srcimg, imgdes* desimg)
{
    int rcode;
    //To view the entire image, use a temporary image descriptor
    // and set the image area to the entire image
    imgdes timage;
    copyimgdes(srcimg, &timage);
    setimagearea(&timage, 0, 0,
        (unsigned)timage.bmh->biWidth - 1,
        (unsigned)timage.bmh->biHeight - 1);

    // If we're displaying a 24-bit image on an 8-bit
    // display adapter, create a temporary 8-bit image for display

```



```

        if (srcimg->bmh->biBitCount == 24)
        {
            rcode = allocimage(desimg, (int)timage.bmh->biWidth,
(int)timage.bmh->biHeight, 8);
            if (rcode == NO_ERROR)
            {
                // Quick dither representation of 24-bit image
                // (Or use colorsscatter)
                rcode = colordither(&timage, desimg, COLORDITHER256);
                rcode = converttrgbtopal(256, &timage, desimg);
                // If error, free allocated memory
                if (rcode != NO_ERROR)
                    freeimage(desimg);
            }
        }
        else
            copyimgdes(&timage, desimg);
        return rcode;
    }
#endif // _VICTOR

/*
void CDIB::ReplaceColor(const int nI, const RGBQUAD& rgbQ)
{
    ASSERT(m_pBMI);
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    ASSERT(m_pBits);

    BYTE* pBits = (BYTE*)GetBitsAddress();
    int iSize = StorageWidth() * DibHeight();
    while (iSize--)
    {
        *pBits = imap[*pBits];
        pBits++;
    }
    LPRGBQUAD pctThis = GetClrTabAddress();
    ASSERT(pctThis);
    RGBQUAD oc = pctThis[nWhat];
    pctThis[nWhat] = pctThis[nWith];
    TRACE("CDIB::ReplaceColor (%d,%d,%d)->(%d,%d,%d)\n",
        oc.rgbRed, oc.rgbGreen, oc.rgbBlue,
        pctThis[nWith].rgbRed, pctThis[nWith].rgbGreen,
        pctThis[nWith].rgbBlue);
}
*/

UINT CDIB::SetPaletteEntries(UINT nStartIndex, UINT nNumEntries,
                            const LPPALETTEENTRY
lpPaletteColors)
{
    RGBQUAD* pctThis = GetClrTabAddress();
    // ::CopyMemory(pctThis, lpPaletteColors + nStartIndex, nNumEntries *
sizeof(PALETTEENTRY));
    // We can't use CopyMemory function, because the byte order of PALETTEENTRY
(R,G,B,f)
    // is different with that of RGBQUAD (B,G,R,res).
    // RGBQUAD      <- when read from the file (byte order reversed in word)

```

```

// PALETTEENTRY <- in memory
for (UINT i=0; i < nNumEntries; i++)
{
    int k = nStartIndex + i;
    pctThis[k].rgbRed      = lpPaletteColors[i].peRed;
    pctThis[k].rgbGreen    = lpPaletteColors[i].peGreen;
    pctThis[k].rgbBlue     = lpPaletteColors[i].peBlue;
    pctThis[k].rgbReserved = lpPaletteColors[i].peFlags;
}
return 0;
}

void CDIB::CopyPaletteEntry(const UINT nDest, const UINT nSrc)
{
    RGBQUAD* pctThis = GetClrTabAddress();
    pctThis[nDest] = pctThis[nSrc];
}
/*
UINT CDIB::SetPaletteEntriesFileIndex(UINT nStartIndex, UINT nNumEntries,
LPPALETTEENTRY lpPaletteColors)
{
    RGBQUAD* pctThis = GetClrTabAddress();
    // ::CopyMemory(pctThis, lpPaletteColors + nStartIndex, nNumEntries *
sizeof(PALETTEENTRY));
    // We can't use CopyMemory function, because the byte order of PALETTEENTRY
(R,G,B,f)
    // is different with that of RGBQUAD (B,G,R,res).
    for (UINT i=0; i < nNumEntries; i++)
    {
        int k = m_imap[nStartIndex + i];
        pctThis[k].rgbRed      = lpPaletteColors[i].peRed;
        pctThis[k].rgbGreen    = lpPaletteColors[i].peGreen;
        pctThis[k].rgbBlue     = lpPaletteColors[i].peBlue;
        pctThis[k].rgbReserved = lpPaletteColors[i].peFlags;
    }
    return 0;
}
*/
UINT CDIB::ShiftRGBPercent(UINT nStartIndex, UINT nNumEntries, int nPer)
{
    RGBQUAD* pctThis = GetClrTabAddress();
    if (nPer < -100)
        nPer = -100;
    float fFactor = (float)nPer / 100;
    for (UINT i=0; i < nNumEntries; i++)
    {
        int k = nStartIndex + i;
        int nVal;
        nVal = pctThis[k].rgbRed + (int)(pctThis[k].rgbRed * fFactor);
        if (nVal > 255)
            nVal = 255;
        pctThis[k].rgbRed      = nVal;
        nVal = pctThis[k].rgbGreen + (int)(pctThis[k].rgbGreen *
fFactor);
        if (nVal > 255)
            nVal = 255;
        pctThis[k].rgbGreen    = nVal;
    }
}

```

```

        nVal = pctThis[k].rgbBlue + (int)(pctThis[k].rgbBlue * fFactor);
        if (nVal > 255)
            nVal = 255;
        pctThis[k].rgbBlue = nVal;
    }
    return 0;
}

UINT CDIB::RotatePaletteIndex(UINT nStartIndex, UINT nNumEntries, UINT nBy)
{
    while (nBy >= nNumEntries)
        nBy -= nNumEntries;
    if (nBy == 0) // original state
        return 0;
    RGBQUAD* pctSt = GetClrTabAddress();
    pctSt += nStartIndex;
    RGBQUAD* pRGBBuf = new RGBQUAD[nNumEntries];
    int nTotalSize = nNumEntries * sizeof(RGBQUAD);
    int nBySize = nBy * sizeof(RGBQUAD);
    ::CopyMemory(pRGBBuf + nBy, pctSt,
        nTotalSize - nBySize);
    ::CopyMemory(pRGBBuf, pctSt + nNumEntries - nBy, nBySize);
    ::CopyMemory(pctSt, pRGBBuf,
        nTotalSize);
    delete [] pRGBBuf;
    return 0;
}
/*
void CDIB::SaveResourceName(LPCSTR szPath)
{
    if (!szPath)
        return;
    int nLen = lstrlen(szPath);
    char* p = strrchr(szPath, '\\'); // Exclude path
    if (!p)
        return;
    char* q = strrchr(szPath, '.'); // Exclude extension
    if (q)
        *q = NULL;
    p++;
    m_strName = (char*)p;
}
*/

```

```

//
//  CDIB:
//
//  (C) Programmed by Kim,
//  UNICHAT INC
//  Information Technology Institute
//
//
//  Nov 25mon '96      Added GIF, TIF, JPG support using Victor Library

// #include <mmsystem.h>      // link winmm.lib
#ifdef _VICTOR
#include <vicdefs.h>      // link Vic32ms.lib, runtime Vic32.dll
#endif

#ifndef __DIB_H
#define __DIB_H

// "DATASRC0|filename.ext"
typedef struct tagDATASOURCEFILEHEADER
{
    WORD  wType;                // 'DS' (0x5344)
    DWORD dwFileSize;
    WORD  wNumEntries;          // Number of entries
    WORD  bcReserved1;
    WORD  bcReserved2;
} DATASOURCEFILEHEADER;

typedef struct tagDATASOURCEENTRY
{
    char  id[13];               // org filename (with extension) + NULL
    DWORD dwOffset;             // offset to the BITMAPFILEHEADER
} DATASOURCEENTRY;

enum IMAGE_OPERATION
{
    OPACITY_100      = 0x0000,    // 100% default
    OPACITY_75       = 0x0001,
    OPACITY_50       = 0x0002,    // 50%
    OPACITY_25       = 0x0004,    // 25%
    OPACITY_12       = 0x0008,    // 12.5% paint
    OPACITY_0        = 0x0010,    //
    OPACITY_MASK     = 0x001F,
    IMAGE_FLIP       = 0x0100,    // change Mirror state
    IMAGE_VERTICAL   = 0x0200,
    ORIENT_MASK      = 0x0F00,
    NO_COLORKEY      = 0x1000,    // Colorkey
    COLORKEY_MASK    = 0xF000,
    DEFAULT_IO       = 0x0000
};

class AFX_EXT_CLASS CDIB : public CObject
{
    DECLARE_SERIAL(CDIB)
public:
    CDIB();

```

```

    // By using virtual destructor, derived classes' destructors can be
    called...
    virtual ~CDIB();
    CDIB& operator=(CDIB& rhs);

// Attributes
    BITMAPINFO*      GetBitmapInfoAddress() const { return m_pBMI; } //
Pointer to bitmap info
    void*            GetBitsAddress() const      { return m_pBits; }
    // Pointer to the bits
    int              GetBitsSize() const;
    RGBQUAD*         GetClrTabAddress() const      // Pointer
to color table
                                { return (LPRGBQUAD)((BYTE*)(m_pBMI)) +
sizeof(BITMAPINFOHEADER)); }
    int              GetNumClrEntries() const;
    // Number of color table entries
    void*            GetPixelAddress(const x, const y) const;
    int              GetBitCount() const          { return m_pBMI->
bmiHeader.biBitCount; }
width
    virtual int      GetWidth() const { return DibWidth(); } // Image
height
    virtual int      GetHeight() const { return DibHeight(); } // Image
    virtual void      GetRect(CRect& rect) const;
    CString*         GetName()              { return &m_strName; }
    enum BITMAP_FILE_TYPE
    {
        FILE_NONE,
        FILE_BMP,
        FILE_BM,      // without palette table
        FILE_BMZ,     // LZ compressed
        FILE_BMC,     // LZ compressed without palette table
        FILE_GIF,
        FILE_TIF,
        FILE_JPG
    };
    BITMAP_FILE_TYPE GetFileType(LPCSTR szFile);

// Operations
    virtual void      Serialize(CArchive& ar);
    BOOL Create(const iWidth, const iHeight, LPCSTR szPalFileName=NULL);
    // Create a new DIB
    BOOL Create(BITMAPINFO* pBMI, BYTE* pBits); // Create
from existing mem
    virtual BOOL      Load(CFile* const fp, LPCSTR szPalFileName=NULL,
                                const BOOL bIsLZ=FALSE,
                                const DWORD dwFileStart=0L); // Load
from file
    virtual BOOL      Load(LPCSTR szFileName=NULL,
                                LPCSTR szPalFileName=NULL); // Load DIB
from disk file
    virtual BOOL      Load(const WORD wResid); //
Load DIB from resource
    BOOL LoadPalette(LPCSTR szPalFileName);
    virtual BOOL      Save(LPCSTR szFileName=NULL); // Save DIB to
disk file

```

```

        virtual BOOL      Save(CFile* const fp, const BOOL bPalette=TRUE);
                               // Save to file
        virtual void      Draw(CDC* pDC, const x, const y);
        void              Draw(CDC* pDC, const xd, const yd, const w, int h,
const xs, const ys);
        void              Tile(CDC* pDC, const x0, const y0, CRect& rcClient);
        virtual BOOL      MapColorsToPalette(const CPalette* pPal);
        virtual void      CopyBits(CDIB* pDIB,
                               const xd, const yd,      const w, int h,
                               const xs, const ys,      const COLORREF

clrTrans=0,
                               const WORD wImOp=DEFAULT_IO);
        void CopyPaletteEntry(const UINT nDest, const UINT nSrc);
        UINT SetPaletteEntries(UINT nStartIndex, UINT nNumEntries,
                               const LPPALETTEENTRY
lpPaletteColors);
        UINT ShiftRGBPercent(UINT nStartIndex, UINT nNumEntries, int nPer);
        UINT RotatePaletteIndex(UINT nStartIndex, UINT nNumEntries, UINT nBy);
        void ClearImage();
        void ClearRect(CRect& rcClear);

protected:
        int              CountBitsSize(const int nW, const int nH) const
                               { return ((nW + 3) & ~3) * nH; }
        BOOL LoadDataSource(LPCSTR szSrcFileName,
                               LPCSTR szFileName,
                               LPCSTR szPalFileName=NULL); // Load DIB
from Data Source
        //      BOOL SaveDataSource(LPCSTR szSrcFileName=NULL,
        //                               LPCSTR szFileName=NULL); //
Save to Data Source

        BITMAPINFO* m_pBMI; // Pointer to BITMAPINFO struct
        BYTE* m_pBits; // Pointer to the bits
        BOOL m_bMyBits; // TRUE if DIB owns Bits memory
        BOOL m_bMapColorsDone;
        CString m_strName; // BMP Filename, resource name

#ifdef _VICTOR
        // For Victor Library
        BOOL LoadFromImage(LPVOID pImg);
        int LoadGIF(LPCSTR fname, imgdes* image);
        int LoadTIF(LPCSTR fname, imgdes* image);
        int LoadJPG(LPCSTR fname, imgdes* image);
        int CreateImage8(imgdes* srcimg, imgdes* desimg);
#endif // _VICTOR

private:
        int DibWidth() const { return m_pBMI->bmiHeader.biWidth;
        }
        int DibHeight() const { return m_pBMI->
>bmiHeader.biHeight; }
        int StorageWidth() const { return (m_pBMI->bmiHeader.biWidth
+ 3) & ~3; }
        };

/*

```

```

// CBmpUsage
class AFX_EXT_CLASS CBmpUsage : public CObject
{
    DECLARE_SERIAL(CBmpUsage)
public:
    CBmpUsage();
    ~CBmpUsage();
    void      AddRef();    { ++m_nRef; }
    int       Release()   { return (--m_nRef); }

protected:
    CString      m_strName;    // Filename
    BITMAPINFO* m_pBMI;        // Pointer to BITMAPINFO struct
    BYTE*        m_pBits;      // Pointer to the bits
    int          m_nRef;        // Reference Count
};

// CBUT : Bitmap Usage Table
class AFX_EXT_CLASS CBUT
{
public:
    CBUT();
    ~CBUT();
};
*/
#endif // __DIB_H

```

```

//
//      CDIBPal :
//
//      (C) Programmed by Kim
//
//      Information Technology Institute
//      UNICHAT INC

#include "stdafx.h"
#include "DIBPal.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CDIBPal

CDIBPal::CDIBPal()
{
}

CDIBPal::~CDIBPal()
{
}

// Create a palette from the color table in a DIB.
BOOL CDIBPal::Create(CDIB* pDIB)
{
    DWORD dwColors = pDIB->GetNumClrEntries();
    // Check the DIB has a color table.
    if (!dwColors)
    {
        TRACE("No color table");
        return FALSE;
    }

    // Get a pointer to the RGB quads in the color table.
    RGBQUAD* pRGB = pDIB->GetClrTabAddress();

    // Allocate a log pal and fill it with the color table info.
    LOGPALETTE* pPal = (LOGPALETTE*)malloc(sizeof(LOGPALETTE) + dwColors *
sizeof(PALETTEENTRY));
    if (!pPal)
    {
        TRACE("Out of memory for logpal");
        return FALSE;
    }
    pPal->palVersion = 0x300; // Windows 3.0
    pPal->palNumEntries = (WORD)dwColors; // Table size
    for (DWORD dw=0; dw < dwColors; dw++)
    {
        pPal->palPalEntry[dw].peRed = pRGB[dw].rgbRed;
        pPal->palPalEntry[dw].peGreen = pRGB[dw].rgbGreen;
        pPal->palPalEntry[dw].peBlue = pRGB[dw].rgbBlue;
        pPal->palPalEntry[dw].peFlags = 0;
    }
}

```



```

    }
    BOOL bResult = CreatePalette(pPal);
    free(pPal);
    return bResult;
}

////////////////////////////////////
// CDIBPal commands

int CDIBPal::GetNumColors()
{
    int iColors = 0;
    if (!GetObject(sizeof(iColors), &iColors))
    {
        TRACE("Failed to get num pal colors");
        return 0;
    }
    return iColors;
}

// bBkgnd: Force Background
void CDIBPal::Draw(CDC* pDC, CRect& rect, const BOOL bBkgnd, const BOOL
bShowIndex)
{
    int iColors = GetNumColors();
    CPalette* pOldPal = pDC->SelectPalette(this, bBkgnd);
    pDC->RealizePalette();
    int i, j, top, left, bottom, right;
    for (j=0, top=0; j<16 && iColors; j++, top=bottom)
    {
        bottom = (j+1) * rect.bottom / 16 + 1;
        for (i=0, left=0; i<16 && iColors; i++, left=right)
        {
            right = (i+1) * rect.right / 16 + 1;
            CBrush br(PALETTEINDEX(j * 16 + i));
            CBrush* brold = pDC->SelectObject(&br);
            pDC->Rectangle(left-1, top-1, right, bottom);
            pDC->SelectObject(brold);
            if (bShowIndex)
            {
                CString strIdx;
                strIdx.Format("%0x", j * 16 + i);
                pDC->SetBkMode(TRANSPARENT);
                pDC->SetTextColor(PALETTEINDEX(128, 128, 0)); //
                Dark Yellow
                pDC->TextOut(left, top, strIdx);
            }
            iColors--;
        }
    }
    pDC->SelectPalette(pOldPal, FALSE);
}

BOOL CDIBPal::SetSysPalColors()
{
    BOOL bResult = FALSE;
    int i, iSysColors, iPalEntries;

```

```

HPALETTE hpalOld;

// Get a screen DC to work with.
HWND hwndActive = ::GetActiveWindow();
HDC hdcScreen = ::GetDC(hwndActive);
ASSERT(hdcScreen);

// Make sure we are on a palettized device.
if (! (::GetDeviceCaps(hdcScreen, RASTERCAPS) & RC_PALETTE))
{
    TRACE("Not a palettized device\n");
    goto abort;
}

// Get the number of system colors and the number of palette
// entries. Note that on a palletized device the number of
// colors is the number of guaranteed colors, i.e., the number
// of reserved system colors.
iSysColors = ::GetDeviceCaps(hdcScreen, NUMCOLORS);
iPalEntries = ::GetDeviceCaps(hdcScreen, SIZEPALETTE);

// If there are more than 256 colors we are wasting our time.
if (iSysColors < 0 || iSysColors > 256) goto abort;

// Now we force the palette manager to reset its tables so that
// the next palette to be realized will get its colors in the order
they are
// in the logical palette. This is done by changing the number of
// reserved colors.
::SetSystemPaletteUse(hdcScreen, SYSPAL_NOSTATIC);
::SetSystemPaletteUse(hdcScreen, SYSPAL_STATIC);

// Select our palette into the screen DC and realize it so that
// its colors will be entered into the free slots in the physical
palette.
hpalOld = ::SelectPalette(hdcScreen, (HPALETTE)m_hObject, // Our hpal
                          FALSE);

#ifdef _DEBUG
    UINT nColorsRemap;
    nColorsRemap = ::RealizePalette(hdcScreen);
    TRACE("CDIBPal::SetSysPalColors() returned %d.\n", nColorsRemap);
#else
    ::RealizePalette(hdcScreen);
#endif

// Now replace the old palette (but don't realize it)
::SelectPalette(hdcScreen, hpalOld, FALSE);

// The physical palette now has our colors set in place and its own
// reserved colors at either end. We can grab the lot now.
PALETTEENTRY pe[256];
GetSystemPaletteEntries(hdcScreen, 0, iPalEntries, pe);

// Set the PC_NOCOLLAPSE flag for each of our colors so that the GDI
// won't merge them. Be careful not to set PC_NOCOLLAPSE for the
// system color entries so that we won't get multiple copies of these
// colors in the palette when we realize it.
for (i = 0; i < iSysColors/2; i++)

```

```

        pe[i].peFlags = 0;
    for (; i < iPalEntries-iSysColors/2; i++)
        pe[i].peFlags = PC_NOCOLLAPSE;
    for (; i < iPalEntries; i++)
        pe[i].peFlags = 0;

    // Resize the palette in case it was smaller.
    ResizePalette(iPalEntries);

    // Update the palette entries with what is now in the physical palette.
    SetPaletteEntries(0, iPalEntries, pe);
    bResult = TRUE;

abort:
    ::ReleaseDC(hwndActive, hdcScreen);
    return bResult;
}

// Load a palette from a named file.
// To get the PALETTEENTRY only, allocate a memory for it and pass it.
BOOL CDIBPal::Load(const char* pszFileName, PALETTEENTRY* pPE)
{
    CString strFile;

    if ((pszFileName == NULL) || (lstrlen(pszFileName) == 0))
    {
        // Show an File Open dialog to get the name.
        CFileDialog dlg (TRUE, // Open
                        NULL, // No default extension
                        NULL, // No initial file name
                        OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
                        "Palette files (*.PAL)|*.PAL|All files
                        (*.*)|*.*|");
        if (dlg.DoModal() == IDOK)
            strFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        strFile = pszFileName;
    }

    // Try to open the file for read access.
    CFile file;
    if (!file.Open(strFile, CFile::modeRead | CFile::shareDenyWrite))
    {
        strFile += " not found!";
        AfxMessageBox(strFile);
        return FALSE;
    }

    BOOL bResult = Load(&file, pPE);
    file.Close();
    if (!bResult)
        AfxMessageBox("Failed to load palette file");
}

```

```

        return bResult;
    }

    // Load a palette from an open CFile object.
    BOOL CDIBPal::Load(CFile* const fp, PALETTEENTRY* pPE)
    {
        return Load(fp->m_hFile, pPE);
    }

    // Load a palette from an open file handle.
    BOOL CDIBPal::Load(const UINT hFile, PALETTEENTRY* pPE)
    {
        MMIOINFO info;
        memset(&info, 0, sizeof(info));
        info.adwInfo[0] = hFile;
        HMMIO hmmio = mmioOpen(NULL, &info, MMIO_READ | MMIO_ALLOCBUF);
        if (!hmmio)
        {
            TRACE("mmioOpen failed\n");
            return FALSE;
        }
        BOOL bResult = Load(hmmio, pPE);
        mmioClose(hmmio, MMIO_FHOPEN);
        return bResult;
    }

    // Load a palette from an open MMIO handle.
    BOOL CDIBPal::Load(const HMMIO hmmio, PALETTEENTRY* pPE)
    {
        // Check whether it's a RIFF PAL file.
        MMCKINFO ckFile;
        ckFile.fccType = mmioFOURCC('P','A','L',' ');
        if (mmioDescend(hmmio, &ckFile, NULL, MMIO_FINDRIFF) != 0)
        {
            TRACE("Not a RIFF or PAL file\n");
            return FALSE;
        }
        // Find the 'data' chunk.
        MMCKINFO ckChunk;
        ckChunk.ckid = mmioFOURCC('d','a','t','a');
        if (mmioDescend(hmmio, &ckChunk, &ckFile, MMIO_FINDCHUNK) != 0)
        {
            TRACE("No data chunk in file\n");
            return FALSE;
        }
        // Allocate some memory for the data chunk.
        int iSize = ckChunk.cksize;
        void* pdata = malloc(iSize);
        if (!pdata)
        {
            TRACE("No mem for data\n");
            return FALSE;
        }
        // Read the data chunk.
        if (mmioRead(hmmio, (char*)pdata, iSize) != iSize)
        {
            TRACE("Failed to read data chunk\n");
        }
    }

```

```

        free(pdata);
        return FALSE;
    }
    // The data chunk should be a LOGPALETTE structure
    // that we can create a palette from.
    LOGPALETTE* pLogPal = (LOGPALETTE*)pdata;
    if (pLogPal->palVersion != 0x300)
    {
        TRACE("Invalid palette version number\n");
        free(pdata);
        return FALSE;
    }
    // Get the number of entries.
    int iColors = pLogPal->palNumEntries;
    if (iColors <= 0)
    {
        TRACE("No colors in palette\n");
        free(pdata);
        return FALSE;
    }
    // If pPE is specified, do not really create the palette
    if (pPE)
    {
        if (iColors > 256)
        {
            TRACE("Colors in this palette exceeds 256: %dn\n",
iColors);
            iColors = 256;
        }
        ::CopyMemory(pPE, &pLogPal->palPalEntry[0],
iColors*sizeof(PALETTEENTRY));
        free(pdata);
        return TRUE;
    }
    else
    {
        return CreatePalette(pLogPal);
    }
}

// Save a palette to an open CFile object.
BOOL CDIBPal::Save(CFile* const fp)
{
    return Save(fp->m_hFile);
}

// Save a palette to an open file handle.
BOOL CDIBPal::Save(const UINT hFile)
{
    MMIOINFO info;
    memset(&info, 0, sizeof(info));
    info.adwInfo[0] = hFile;
    HMMIO hmmio = mmioOpen(NULL, &info, MMIO_WRITE | MMIO_CREATE |
MMIO_ALLOCBUF);
    if (!hmmio)
    {
        TRACE("mmioOpen failed\n");
    }

```

```

        return FALSE;
    }
    BOOL bResult = Save(hmmio);
    mmioClose(hmmio, MMIO_FHOPEN);
    return bResult;
}

// Save a palette to an open MMIO handle.
BOOL CDIBPal::Save(const HMMIO hmmio)
{
    // Create a RIFF chunk for a PAL file.
    MMCKINFO ckFile;
    ckFile.cksize = 0; // Corrected later
    ckFile.fccType = mmioFOURCC('P','A','L',' ');
    if (mmioCreateChunk(hmmio, &ckFile, MMIO_CREATERIFF) != 0)
    {
        TRACE("Failed to create RIFF-PAL chunk\n");
        return FALSE;
    }
    // Create the LOGPALETTE data which will become the data chunk.
    int iColors = GetNumColors();
    ASSERT(iColors > 0);
    int iSize = sizeof(LOGPALETTE) + (iColors-1) * sizeof(PALETTEENTRY);
    LOGPALETTE* plp = (LOGPALETTE*)malloc(iSize);
    ASSERT(plp);
    plp->palVersion = 0x300;
    plp->palNumEntries = iColors;
    GetPaletteEntries(0, iColors, plp->palPalEntry);
    // create the data chunk.
    MMCKINFO ckData;
    ckData.cksize = iSize;
    ckData.ckid = mmioFOURCC('d','a','t','a');
    if (mmioCreateChunk(hmmio, &ckData, 0) != 0)
    {
        TRACE("Failed to create data chunk\n");
        return FALSE;
    }
    // Write the data chunk.
    if (mmioWrite(hmmio, (char*)plp, iSize) != iSize)
    {
        TRACE("Failed to write data chunk\n");
        free(plp);
        return FALSE;
    }
    free(plp);
    // Ascend from the data chunk which will correct the length.
    mmioAscend(hmmio, &ckData, 0);
    // Ascend from the RIFF-PAL chunk.
    mmioAscend(hmmio, &ckFile, 0);

    return TRUE;
}

```

```

//
//      CDIBPal
//
//      (C) Programmed by Kim
//
//      Information Technology Institute
//      UNICHAT INC
//

#ifndef __DIBPAL__
#define __DIBPAL__

#include "DIB.h"
#include <mmsystem.h>

class AFX_EXT_CLASS CDIBPal : public CPalette
{
public:
    CDIBPal();
    virtual ~CDIBPal();
    BOOL Create(CDIB* pDIB);          // Create from a DIB
    int      GetNumColors();          // Get the number of colors in the
palette.
    void Draw(CDC* pDC, CRect& rect, const BOOL bBgnd=FALSE, const BOOL
bShowIndex=FALSE);
    BOOL SetSysPalColors();
    BOOL Load(const char* pszFileName=NULL, PALETTEENTRY* pPE=NULL);
    BOOL Load(CFile* const fp, PALETTEENTRY* pPE=NULL);
    BOOL Load(const UINT hFile, PALETTEENTRY* pPE=NULL);
    BOOL Load(const HMMIO hmmio, PALETTEENTRY* pPE=NULL);
    BOOL Save(CFile* const fp);
    BOOL Save(const UINT hFile);
    BOOL Save(const HMMIO hmmio);
    int      GetEntries(PALETTEENTRY* pPE);
};

#endif // __DIBPAL__

```

UC2Ani\Make.txt

LINK: winmm.lib Vic32ms.lib
Runtime DLL: Vic32.dll


```

//
//  CMCIObject :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "mciobj.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CMCIObject

IMPLEMENT_SERIAL(CMCIObject, CObject, 0 /* schema number*/ )

CMCIObject::CMCIObject()
{
    m_OpenParams.dwCallback = 0;
    m_OpenParams.wDeviceID = 0;
    m_OpenParams.lpstrDeviceType = NULL;
    m_OpenParams.lpstrElementName = NULL;
    m_OpenParams.lpstrAlias = NULL;
}

CMCIObject::~CMCIObject()
{
    // make sure the object is not in use
    if (m_OpenParams.wDeviceID)
        Close();
    ASSERT(m_OpenParams.wDeviceID == 0);
}

////////////////////////////////////
// CMCIObject serialization

void CMCIObject::Serialize(CArchive& ar)
{
    ASSERT(1); // not supported
}

////////////////////////////////////
// CMCIObject commands

void CMCIObject::MCIError(DWORD dwErr)
{
    char buf[256];

    buf[0] = '\0';
    mciGetErrorString(dwErr, buf, sizeof(buf));
}

```

```

        if (!strlen(buf))
            strcpy(buf, "Unknown error");
        TRACE(buf);
    }

    BOOL CMCIObject::Load(const char* pszFileName)
    {
        DWORD dwResult;

        if (m_OpenParams.wDeviceID)
            Close();
        ASSERT(m_OpenParams.wDeviceID == 0);
        m_OpenParams.lpstrDeviceType = NULL;
        m_OpenParams.lpstrElementName = pszFileName;
        dwResult = mciSendCommand(0,
                                   MCI_OPEN, MCI_WAIT | MCI_OPEN_ELEMENT,
                                   (DWORD) (LPVOID) &m_OpenParams);

        if (dwResult)
        {
            MCIError(dwResult);
            m_OpenParams.wDeviceID = 0;
            return FALSE;
        }

        // Set the time format to milliseconds
        MCI_SET_PARMS set;
        set.dwTimeFormat = MCI_FORMAT_MILLISECONDS;
        dwResult = mciSendCommand(m_OpenParams.wDeviceID,
                                   MCI_SET, MCI_WAIT | MCI_SET_TIME_FORMAT,
                                   (DWORD) (LPVOID) &set);

        if (dwResult)
        {
            MCIError(dwResult);
            m_OpenParams.wDeviceID = 0;
            return FALSE;
        }

        CString strName(pszFileName);
        strName.MakeLower();
        int i = strName.ReverseFind('.');
        if (strName[i+1] != 'm') // only for wave
        {
            // Cue the file so it will play with no delay
            dwResult = mciSendCommand(m_OpenParams.wDeviceID,
                                       MCI_CUE, MCI_WAIT,
                                       (DWORD) (LPVOID) NULL);

            if (dwResult)
            {
                MCIError(dwResult);
                m_OpenParams.wDeviceID = 0;
                return FALSE;
            }
        }
        return TRUE;
    }

    BOOL CMCIObject::OpenDevice(const char* pszDevName)

```

```

{
    DWORD dwResult;

    if (m_OpenParams.wDeviceID)
        Close();
    ASSERT(m_OpenParams.wDeviceID == 0);
    m_OpenParams.lpstrDeviceType = pszDevName;
    dwResult = mciSendCommand(0,
                                MCI_OPEN, MCI_WAIT | MCI_OPEN_SHAREABLE |
MCI_OPEN_TYPE,
                                (DWORD) (LPVOID) &m_OpenParams);

    if (dwResult)
    {
        MCIError(dwResult);
        m_OpenParams.wDeviceID = 0;
        return FALSE;
    }

    // Set the time format to milliseconds
    MCI_SET_PARMS set;
    set.dwTimeFormat = MCI_FORMAT_MILLISECONDS;
    dwResult = mciSendCommand(m_OpenParams.wDeviceID,
                                MCI_SET, MCI_WAIT | MCI_SET_TIME_FORMAT,
                                (DWORD) (LPVOID) &set);

    if (dwResult)
    {
        MCIError(dwResult);
        m_OpenParams.wDeviceID = 0;
        return FALSE;
    }
    return TRUE;
}

void CMCIObject::Close()
{
    MCI_GENERIC_PARMS gp;
    DWORD dwResult;

    if (m_OpenParams.wDeviceID == 0)
        return; // already closed
    Stop(); // just in case
    dwResult = mciSendCommand(m_OpenParams.wDeviceID,
                                MCI_CLOSE, MCI_WAIT,
                                (DWORD) (LPVOID) &gp);

    if (dwResult)
        MCIError(dwResult);
    m_OpenParams.wDeviceID = 0;
}

BOOL CMCIObject::Play()
{
    MCI_PLAY_PARMS play;
    DWORD dwResult;

    if (m_OpenParams.wDeviceID == 0)
        return FALSE; // not open

```

```

        mciSendCommand(m_OpenParams.wDeviceID,
                        MCI_SEEK, MCI_WAIT | MCI_SEEK_TO_START,
                        0);
        dwResult = mciSendCommand(m_OpenParams.wDeviceID,
                                   MCI_PLAY, NULL,
                                   (DWORD) (LPVOID)&play);

        if (dwResult)
        {
            MCIError(dwResult);
            return FALSE;
        }
        return TRUE;
    }

void CMCIObject::Stop()
{
    DWORD dwResult;

    if (m_OpenParams.wDeviceID == 0)
        return; // not open

    dwResult = mciSendCommand(m_OpenParams.wDeviceID,
                               MCI_STOP, MCI_WAIT,
                               (DWORD) (LPVOID) NULL);

    if (dwResult)
        MCIError(dwResult);
}

DWORD CMCIObject::GetPosition()
{
    if (m_OpenParams.wDeviceID == 0)
        return 0; // not open

    MCI_STATUS_PARMS status;
    status.dwItem = MCI_STATUS_POSITION;
    if (mciSendCommand(m_OpenParams.wDeviceID,
                        MCI_STATUS, MCI_WAIT | MCI_STATUS_ITEM,
                        (DWORD) (LPVOID)&status) != 0)
    {
        return 0; // some error
    }
    return status.dwReturn;
}

```

```

//
//  CMCIObj
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
#ifndef __MCIOBJ__
#define __MCIOBJ__

#include <mmsystem.h>

////////////////////////////////////
// CMCIObject object

class AFX_EXT_CLASS CMCIObject : public CObject
{
    DECLARE_SERIAL(CMCIObject)
public:
    CMCIObject();
    ~CMCIObject();
    BOOL Load(const char* pszFileName);
    BOOL OpenDevice(const char* pszDevName);
    void Close();
    BOOL Play();
    void Stop();
    DWORD GetPosition();

// Implementation
protected:
    virtual void Serialize(CArchive& ar); // Overridden for document i/o

private:
    void MCIError(DWORD dwErr);

    MCI_OPEN_PARMS m_OpenParams;
};

#endif // __MCIOBJ__

```

```

//
//  COSBView: Off-Screen Buffered View
//
//  (C) Programmed by KEN
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "DIB.h"
#include "DIBPal.h"
#include "OSBView.h"
#include "PhSprite.h"    // AddDirtyRegion(CPhasedSprite* pPS);
#include "Bubble.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
// #if !defined(KSM_REPORT)
// #define KSM_REPORT
// #endif
#endif

////////////////////////////////////
// COSBView

IMPLEMENT_DYNCREATE(COSBView, CScrollView)

BEGIN_MESSAGE_MAP(COSBView, CScrollView)
    //{AFX_MSG_MAP(COSBView)
    ON_WM_PALETTECHANGED()
    ON_WM_QUERYNEWPALETTE()
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// COSBView construction/destruction

COSBView::COSBView()
{
    m_pOSB                = NULL;
    m_pOSBPal              = NULL;
    m_hDIBSection          = NULL;
    m_pOneToOneClrTab      = NULL;

    BOOL bUseCreatedDIBSection = FALSE;
    // See if we are on Win32s which exports CreatedDIBSection but does not
    support it
    DWORD dwVer = ::GetVersion();
    if ((dwVer & 0x800000FF) != 0x08000003)
    {
        // Not on Win32s so try to get the CreatedDIBSection procedure
        address.
        HANDLE hMod = ::GetModuleHandle("gdi32");
        if (hMod)
        {

```

```

        m_pCreatedDIBSection =
(CDSPROC*)GetProcAddress((HMODULE)hMod, "CreatedDIBSection");
        if (m_pCreatedDIBSection)
            bUseCreatedDIBSection = TRUE;
    }
    if (!bUseCreatedDIBSection)
    {
        AfxMessageBox("Sorry, This Program is only for Windows 95 and
NT.");
        PostMessage(WM_CLOSE);
    }
}

COSBView::~COSBView()
{
    if (m_posb) delete m_posb;
    if (m_posbPal) delete m_posbPal;
    if (m_pOneToOneClrTab) free(m_pOneToOneClrTab);
    if (m_hDIBSection) ::DeleteObject(m_hDIBSection);
    EmptyDirtyList();
}

// Create a new buffer, tables and palette to match a supplied DIB.
BOOL COSBView::CreateOSB(CDIB* pDIB)
{
    TRACE("COSBView::CreateOSB(%lx)\n", pDIB);
    // Create the 1-to-1 palette index table.
    if (m_pOneToOneClrTab)
        free(m_pOneToOneClrTab);
    m_pOneToOneClrTab = (LPBITMAPINFO)malloc(sizeof(BITMAPINFOHEADER) + 256
* sizeof(WORD));
    if (!m_pOneToOneClrTab)
    {
        TRACE("Failed to create color table\n");
        return FALSE;
    }

    // Set up the table header to match the DIB by copying the header
    // and then constructing the 1-to-1 index translation table.
    ::CopyMemory(m_pOneToOneClrTab, pDIB->GetBitmapInfoAddress(),
sizeof(BITMAPINFOHEADER));
    // make sure the length of the table is set to 256 not to
    // the number of colors in the DIB which is irrelevant
    m_pOneToOneClrTab->bmiHeader.biClrUsed = 0; // default (256)
    WORD* pIndex = (LPWORD)((LPBYTE)m_pOneToOneClrTab +
sizeof(BITMAPINFOHEADER));
    for (int i = 0; i < 256; i++)
        *pIndex++ = (WORD)i;

    // Create a palette from the DIB so that we can use it to do screen
    drawing.
    if (m_posbPal)
        delete m_posbPal;
    m_posbPal = new CDIBPal;
    ASSERT(m_posbPal);
    if (m_posbPal->Create(pDIB))

```

```

    {
        // Map the colors so that we get an identity palette.
        m_posBPal->SetSysPalColors();
    }
    else
    {
        TRACE("Failed to create palette\n");
        delete m_posBPal;
        m_posBPal = NULL;
        return FALSE;
    }

    // Delete any existing DIB and create a new one.
    if (m_posB)
        delete m_posB;
    m_posB = new CDIB;
    BOOL bResult = FALSE;
    // if (m_bUseCreateDIBSection)
    // {
    if (m_hDIBSection)
        ::DeleteObject(m_hDIBSection);
    ASSERT(m_pCreateDIBSection);
    CDC* pDC = GetDC();
    CPalette* pPalOld = pDC->SelectPalette(m_posBPal, FALSE);    //
foreground
    pDC->RealizePalette();
    BYTE* pBits = NULL;
    // The CreateDIBSection function creates a device-independent bitmap
(DIB)
    // that applications can write to directly.
    m_hDIBSection = (*m_pCreateDIBSection)(pDC->GetSafeHdc(),    //
CreatedDIBSection
        m_pOneToOneClrTab, DIB_PAL_COLORS, (VOID**)&pBits,
NULL, 0);
    pDC->SelectPalette(pPalOld, FALSE);
    ASSERT(m_hDIBSection);
    ASSERT(pBits);
    ReleaseDC(pDC);    // int CWnd::ReleaseDC(CDC* pDC);
    bResult = m_posB->Create(pDIB->GetBitmapInfoAddress(), pBits);
    // }
    // else
    // {
    //     bResult = m_posB->Create(pDIB->GetWidth(), pDIB->GetHeight());
    // }
    if (!bResult)
    {
        TRACE("Failed to create off-screen DIB\n");
        delete m_posB;
        m_posB = NULL;
        return FALSE;
    }

    CSize sizeTotal(m_posB->GetWidth(), m_posB->GetHeight());
    SetScrollSizes(MM_TEXT, sizeTotal);

    return TRUE;
}

```



```

////////////////////////////////////
// COSBView drawing

```

```

void COSBView::OnInitialUpdate()
{
    CSize sizeTotal;
    if (m_pOSB)
    {
        sizeTotal.cx = m_pOSB->GetWidth();
        sizeTotal.cy = m_pOSB->GetHeight();
    }
    else
    {
        sizeTotal.cx = 640;
        sizeTotal.cy = 480;
    }
    SetScrollSizes(MM_TEXT, sizeTotal);
    CScrollView::OnInitialUpdate();
}

```

```

// Updated Feb 16 '98 Soomin, Kim

```

```

void COSBView::OnDraw(CDC* pDC)
{
    // CView::OnDraw(pDC);      // OnPrepareDC
    CRect rc;    // Invalid Rect
    pDC->GetClipBox(rc);
    TRACE("COSBView::OnDraw(%d,%d,%d,%d) W:%d,H:%d\n",
        rc.left, rc.top, rc.right, rc.bottom, rc.Width(),
        rc.Height());
    DrawOSB(&rc);
}

```

```

////////////////////////////////////
// COSBView diagnostics

```

```

#ifdef _DEBUG
void COSBView::AssertValid() const
{
    CScrollView::AssertValid();
}

```

```

void COSBView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

```

```

CDocument* COSBView::GetDocument() // Non-debug version is inline
{
    return m_pDocument;
}

```

```

#endif // _DEBUG

```

```

////////////////////////////////////
// COSBView message handlers

```

```

// Draw a section of the off-screen image buffer to the screen.

```

```

void COSBView::DrawOSB(CRect* pClipRect)
{
    CClientDC dc(this);
    OnPrepareDC(&dc); // CScroll changes the viewport origin and mapping
mode.    CRect rcDraw;

    // Make sure we have what we need to do a paint.
    if (!m_pOSB || !m_pOneToOneClrTab)
    {
        TRACE("COSBView::DrawOSB() - No DIB or color table to paint
from.\n");
        return;
    }
    // See if a clip rect was supplied, and use the client area if not.
    if (pClipRect)
    {
        rcDraw = *pClipRect;
    }
    else
    {
        GetClientRect(rcDraw);
        dc.DPtoLP(&rcDraw); // THIS WAS THE KEY!
    }

    // Get the clip box.
    CRect rcClip;
    dc.GetClipBox(rcClip);

    // Create a rect for the DIB.
    CRect rcDIB(0, 0, m_pOSB->GetWidth()-1, m_pOSB->GetHeight()-1);

    // Find a rectangle that describes the intersection of the draw rect,
    // clip rect, and DIB rect.
    CRect rcBlt = rcDraw & rcClip & rcDIB;

    // Copy the update rectangle from the off-screen DC to the window DC.
    // Note that the DIB origin is the bottom-left corner.
    int w = rcBlt.right - rcBlt.left;
    int h = rcBlt.bottom - rcBlt.top;
    if (w == 0 && h == 0)
        return; // nothing to draw
    int xs, ys, xd, yd;
    xs = xd = rcBlt.left;
    // ys = (m_bUseCreatedIBSection) ? rcBlt.top : m_pOSB->GetHeight() -
rcBlt.bottom;
    ys = yd = rcBlt.top;

    // If we have a palette, select and realize it.
    CPalette* ppalOld = NULL;
    if (m_pOSBPal)
    {
        ppalOld = dc.SelectPalette(m_pOSBPal, FALSE); // foreground
        dc.RealizePalette();
    }

    // if (m_bUseCreatedIBSection)

```

```

// {
HDC dcMem = ::CreateCompatibleDC(dc.GetSafeHdc());
HBITMAP hbmOld = (HBITMAP)::SelectObject(dcMem, m_hDIBSection);

// Note: you do not need to select the palette into the memory DC
// because the DIB section is using palette index values not colors.
::BitBlt(dc.GetSafeHdc(), xd, yd, w, h, dcMem, xs, ys, SRCCOPY);
::SelectObject(dcMem, hbmOld);
::DeleteDC(dcMem);

// CDC dcMem;
// dcMem.CreateCompatibleDC(&dc); // pass NULL to be compatible
with system display
// CBitmap* bmpOld =
dcMem.SelectObject(CBitmap::FromHandle(m_hDIBSection));
// dc.BitBlt(xd, yd, w, h, &dcMem, xs, ys, SRCCOPY);
// dcMem.SelectObject(bmpOld);
// dcMem.DeleteDC();
/* }
else
{
    BYTE* pBits = (BYTE*)m_posb->GetBitsAddress();
    ::StretchDIBits(dc.GetSafeHdc(),
                    xd, yd, w, h, // Destination x, y,
width, height xs, ys, w, h, // Source x, y,
width, height pBits, // Pointer to bits
m_pOneToOneClrTab, // BITMAPINFO
DIB_PAL_COLORS, // Options
SRCCOPY); // ROP
}
*/
// Select old palette if we altered it.
if (ppalOld)
    dc.SelectPalette(ppalOld, FALSE);
#ifdef _DEBUG_RENDER
static int D_DrawCount = 0;
if (D_DrawCount++ % 100 == 0)
    TRACE3("COSBView::Draw(W:%d,H:%d)=%d times\n", w, h,
D_DrawCount);
#endif
}

// Add a region to the dirty list.
void COSBView::AddDirtyRegion(CRect* prcNew)
{
    // Get the rectangle currently at the top of the list.
    POSITION pos = m_DirtyList.GetHeadPosition();
    if (pos)
    {
        CRect* prcTop = (CRect*)m_DirtyList.GetNext(pos);
        CRect rcTest;
        // If the new one intersects the top one merge them.
        if (rcTest.IntersectRect(prcTop, prcNew))
        {
            prcTop->UnionRect(prcTop, prcNew);
            return;
        }
    }
}

```

```

    }
}
// List is empty, or there was no intersection.
CRect* prc = new CRect;
*prc = *prcNew; // Copy the data.
// Add a new rectangle to the list.
m_DirtyList.AddHead((CObject*)prc);
}

void COSBView::AddDirtyRegion(CPhasedSprite* pPS)
{
    // if (!pPS)
    //     return;
    CRect rcDirty;
    pPS->GetRect(rcDirty);
    AddDirtyRegion(&rcDirty);
}

void COSBView::AddDirtyRegion(CBubble* pBB)
{
    CRect rcDirty;
    pBB->GetRect(rcDirty);
    AddDirtyRegion(&rcDirty);
}

// Render and draw all the dirty regions.
void COSBView::RenderAndDrawDirtyList()
{
    POSITION pos = m_DirtyList.GetHeadPosition();
    // Render all the dirty regions.
    while (pos)
    {
        // Get the next region.
        CRect* pRect = (CRect*)m_DirtyList.GetNext(pos);
        // Render it.
        Render(pRect);
    }
    // Draw all the dirty regions to the screen.
    while (!m_DirtyList.IsEmpty())
    {
        // Get the next region.
        CRect* pRect = (CRect*)m_DirtyList.RemoveHead();
        DrawOSB(pRect);
        // InvalidateRect(pRect, FALSE); // Convert to LP
        // Done with it.
        delete pRect;
    }
}

// Empty the dirty list.
void COSBView::EmptyDirtyList()
{
    while (!m_DirtyList.IsEmpty())
    {
        CRect* prc = (CRect*)m_DirtyList.RemoveHead();
        delete prc;
    }
}

```

```

}

// Update the view to reflect some change in the doc.
void COSBView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    // Render and draw everything.
    Render();
    // DrawOSB();
    Invalidate(FALSE);
}

////////////////////////////////////
// Palette Control

void COSBView::OnPaletteChanged(CWnd* pFocusWnd)
{
    TRACE0("COSBView::OnPaletteChanged\n");
    // See if the change was caused by us and ignore it if not.
    if (pFocusWnd != this)
        OnQueryNewPalette();
}

// Note: Windows actually ignores the return value.
BOOL COSBView::OnQueryNewPalette()
{
    TRACE0("COSBView::OnQueryNewPalette\n");
    // We are going active so realize our palette.
    if (m_posBPal)
    {
        CDC* pdc = GetDC();
        CPalette* poldpal = pdc->SelectPalette(m_posBPal, FALSE);    //
foreground
        UINT u = pdc->RealizePalette();
        if (poldpal)
            pdc->SelectPalette(poldpal, FALSE);
        ReleaseDC(pdc);
        if (u)
        {
            // Some colors changed so we need to do a repaint.
            Invalidate(); // Repaint the lot.
            return TRUE; // Say we did something.
        }
    }
    return FALSE; // Say we did nothing.
}

```

```

//
//  COSBView:
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
////////////////////////////////////

#ifndef __OSBVIEW_H
#define __OSBVIEW_H

class CDIB;
class CDIBPal;
class CPhasedSprite;
class CBubble;

// Define the CreateDIBSection function.
typedef HBITMAP (APIENTRY CDSPROC)
                (HDC hdc, BITMAPINFO* pbmi, UINT iUsage,
                 VOID** ppvBits, HANDLE hSection, DWORD dwOffset);

class AFX_EXT_CLASS COSBView : public CScrollView
{
protected: // create from serialization only
    COSBView();
    DECLARE_DYNCREATE(COSBView)

// Attributes
public:
    CDocument*   GetDocument();
    CDIB*        GetOSB()           { return m_pOSB; }
    CDIBPal*     GetOSBPalette()    { return m_pOSBPal; }
    HBITMAP      GetHBitmap() const { return m_hDIBSection; }

// Operations
public:
    virtual BOOL      CreateOSB(CDIB* pDIB);           // Create a new
buffer
    void             DrawOSB(CRect* pClipRect=NULL);   // Draw off-
screen buffer to screen
    virtual void      Render(CRect* pClipRect=NULL) { return; }
    void             AddDirtyRegion(CRect* pRect);
    void             AddDirtyRegion(CPhasedSprite* pPS);
    void             AddDirtyRegion(CBubble* pBB);
    void             RenderAndDrawDirtyList();

// Implementation
public:
    virtual ~COSBView();
    virtual void      OnDraw(CDC* pDC); // Overridden to draw this view.
    virtual void      OnInitialUpdate(); // First time after
construction.
    virtual void      OnUpdate(CView* pSender, LPARAM lHint, CObject*
pHint);

#ifdef _DEBUG

```

```

        virtual void      AssertValid() const;
        virtual void      Dump(CDumpContext& dc) const;
    #endif

protected:
        CDIB*             m_pOSB;                // The DIB buffer.
        CDIBPal*          m_pOSBPal;             // Palette for drawing.
        HBITMAP           m_hDIBSection;         // Bitmap from CreateDIBSection

private:
        BITMAPINFO* m_pOneToOneClrTab;           // Pointer to 1-to-1 color
        table
        //  BOOL          m_bUseCreateDIBSection; // Flag
        CDSPROC*        m_pCreateDIBSection;     // Pointer to CreateDIBSection
        COBList          m_DirtyList;            // Dirty regions

        void EmptyDirtyList();

// Generated message map functions
protected:
        //{AFX_MSG(COSBView)
        afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
        afx_msg BOOL OnQueryNewPalette();
        //}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG // Debug version in osbview.cpp
inline CDocument* COSBView::GetDocument()
{ return (CDocument*) m_pDocument; }
#endif

////////////////////////////////////
#endif // __OSBVIEW_H

```

```

//
//  CPhasedSprite
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "dib.h"
#include "spriteno.h"
#include "sprite.h"
#include "phsprite.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

const DWORD LONGTIMELATER = 3600000L;

////////////////////////////////////
// CPhasedSprite

IMPLEMENT_SERIAL(CPhasedSprite, CSprite, 0)

CPhasedSprite::CPhasedSprite()
{
    m_iNumCellRows      = m_iNumCellColumns      = 1;
    m_iCellRow          = m_iCellColumn          = 0;
    m_wType              = SPRITE_PHASED;
    m_ai                = 0; // animation index
    m_dwAlarmTick        = ::GetTickCount() + LONGTIMELATER; // Never call
    HeartBeat before Load
    m_nMSPT              = 0;
    m_strHLink.Empty();
}

CPhasedSprite::~CPhasedSprite()
{
}

CPhasedSprite& CPhasedSprite::operator=(const CPhasedSprite& rhs)
{
    if (this == &rhs)
        return *this;
    *(CSprite*)this = *((CSprite*)&rhs);
    m_iNumCellRows      = rhs.m_iNumCellRows;
    m_iNumCellColumns    = rhs.m_iNumCellColumns;
    m_iCellRow           = rhs.m_iCellRow;
    m_iCellColumn        = rhs.m_iCellColumn;
    m_iCellHeight        = rhs.m_iCellHeight;
    m_iCellWidth         = rhs.m_iCellWidth;
    m_ai                = rhs.m_ai;
    m_nMSPT              = rhs.m_nMSPT;
    m_dwAlarmTick        = rhs.m_dwAlarmTick;
}

```



```

        m_strHLink          = rhs.m_strHLink;
    return *this;
}

BOOL CPhasedSprite::operator==(const CPhasedSprite& rhs)
{
    return ((*CSprite*)this == *((CSprite*)&rhs)) &&
        (GetCellID() == rhs.GetCellID()) &&
        (m_ai == rhs.m_ai));
}

/////////////////////////////////////////////////////////////////
// CPhasedSprite serialization

void CPhasedSprite::Serialize(CArchive& ar)
{
    CSprite::Serialize(ar);
    if (ar.IsStoring())
    {
        ar << (DWORD) m_iNumCellRows;
        ar << (DWORD) m_iNumCellColumns;
        ar << (DWORD) m_iCellRow;
        ar << (DWORD) m_iCellColumn;
    }
    else
    {
        DWORD dw;
        ar >> dw;
        SetNumCellRows(dw);
        ar >> dw;
        SetNumCellColumns(dw);
        ar >> dw;
        SetCellRow(dw);
        ar >> dw;
        SetCellColumn(dw);
    }
}

/////////////////////////////////////////////////////////////////
// CPhasedSprite commands

// Do any initialization after file load of a new image etc.
void CPhasedSprite::Initialize()
{
    CSprite::Initialize();
    m_iNumCellRows      = m_iNumCellColumns      = 1;
    m_iCellRow          = m_iCellColumn          = 0;
    m_iCellHeight       = CSprite::GetHeight();
    m_iCellWidth        = CSprite::GetWidth();
}

// Divide the image into a given number of rows.
BOOL CPhasedSprite::SetNumCellRows(const int iRows)
{
    if (iRows < 1)
    {
        TRACE1("Invalid number of rows(%d)\n", iRows);
    }
}

```

```

        return FALSE;
    }
    // Compute the height of each row.
    int iCellHeight = CSprite::GetHeight() / iRows;
    if (iCellHeight < 1)
    {
        TRACE0("Can't make them that small.\n");
        return FALSE;
    }
    // Set the new height and row count.
    m_iNumCellRows = iRows;
    m_iCellRow = 0;
    m_iCellHeight = iCellHeight;
    return TRUE;
}

// Divide the image into a given number of columns.
BOOL CPhasedSprite::SetNumCellColumns(const int iColumns)
{
    if (iColumns < 1)
    {
        TRACE1("Invalid number of columns(%d)\n", iColumns);
        return FALSE;
    }
    // Compute the width of each column.
    int iCellWidth = CSprite::GetWidth() / iColumns;
    if (iCellWidth < 1)
    {
        TRACE("Can't make them that small.\n");
        return FALSE;
    }
    // Set the new width and column count.
    m_iNumCellColumns = iColumns;
    m_iCellColumn = 0;
    m_iCellWidth = iCellWidth;
    return TRUE;
}

BOOL CPhasedSprite::SetNumCells(const int iRows, const int iColumns)
{
    SetNumCellRows(iRows);
    return (SetNumCellColumns(iColumns));
}

// Set the current column.
BOOL CPhasedSprite::SetCellColumn(const int iColumn)
{
    if ((iColumn >= m_iNumCellColumns) || (iColumn < 0))
    {
        TRACE1("Invalid column(%d)\n", iColumn);
        return FALSE;
    }
    if (iColumn == m_iCellColumn) return FALSE; // Nothing to do
    m_iCellColumn = iColumn;
    // Send a notification to redraw.
    if (m_pNotifyObj)
    {

```

```

        CRect rcPos;
        GetRect(rcPos);
        m_pNotifyObj->Change(this, CSpriteNotifyObj::IMAGE, &rcPos);
    }
    return TRUE;
}

// Set the current row.
BOOL CPhasedSprite::SetCellRow(const int iRow)
{
    if ((iRow >= m_iNumCellRows) || (iRow < 0))
    {
        TRACE("%x CPhasedSprite::SetCellRow - Invalid row(%d)\n", this,
iRow);
        return FALSE;
    }
    if (iRow == m_iCellRow) return FALSE; // Nothing to do
    m_iCellRow = iRow;
    // Send a notification to redraw.
    if (m_pNotifyObj)
    {
        CRect rcPos;
        GetRect(rcPos);
        m_pNotifyObj->Change(this, CSpriteNotifyObj::IMAGE, &rcPos);
    }
    return TRUE;
}

BOOL CPhasedSprite::SetCell(const int iRow, const int iColumn)
{
    if (iRow == m_iCellRow && iColumn == m_iCellColumn)
        return FALSE; // Nothing to do
    if ((iRow >= m_iNumCellRows) || (iRow < 0))
    {
        TRACE("%x CPhasedSprite::SetCell - Invalid row(%d)\n", this,
iRow);
        return FALSE;
    }
    m_iCellRow = iRow;
    if ((iColumn >= m_iNumCellColumns) || (iColumn < 0))
    {
        TRACE("%x CPhasedSprite::SetCell - Invalid column(%d)\n", this,
iColumn);
        return FALSE;
    }
    m_iCellColumn = iColumn;
    // Send a notification to redraw.
    if (m_pNotifyObj)
    {
        CRect rcPos;
        GetRect(rcPos);
        m_pNotifyObj->Change(this, CSpriteNotifyObj::IMAGE, &rcPos);
    }
    return TRUE;
}

BOOL CPhasedSprite::SetNextCell()

```

```

{
    int nID = GetCellID();
    nID++;
    if (nID >= GetNumCells())
        nID = 0;
    return SetCell(nID);
}

// Test for a hit in a non-transparent area.
BOOL CPhasedSprite::HitTest(const CPoint& point) const
{
    // Test if the point is inside the sprite rectangle.
    if ((point.x > m_x) && (point.x < m_x + GetWidth()) &&
        (point.y > m_y) && (point.y < m_y + GetHeight()))
    {
        // Hit is in sprite rectangle.
        // See if this point is transparent by testing if the pixel value
        // is the same as the top-left corner value.
        // Note that top-left of the image is bottom-left in the DIB.
        // Get the address of the top-left pixel of the cell, not the
        DIB.
        // Note that the GetPixelAddress function refers to addresses in
        the DIB, not the cell.
        int x = point.x - m_x + (m_iCellColumn * m_iCellWidth);
        int y = point.y - m_y + (m_iCellRow * m_iCellHeight);
        BYTE* p = (BYTE*)m_pDIB->GetPixelAddress(x, y);
        ASSERT(p);
        if (*p != m_btTransIndex)
            return TRUE;
    }
    return FALSE;
}

// Render a sprite to a DIB (usually an OSB).
void CPhasedSprite::Render(CDIB* pDIB, const CRect* pClipRect)
{
    ASSERT(pDIB);
    // Get the sprite rect and see if it's visible.
    CRect rcThis; // This sprite's region
    GetRect(rcThis); // CSprite::GetRect {m_x, m_y,
m_x+GetWidth(), m_y+GetHeight()}
    CRect rcDraw;
    if (pClipRect)
    {
        if (!rcDraw.IntersectRect(pClipRect, &rcThis))
            return; // Not visible
    }
    else
    {
        if (!pClipRect || (m_wImOp & OPACITY_MASK))
        {
            CRect rcOSB;
            pDIB->GetRect(rcOSB);
            rcDraw.IntersectRect(&rcOSB, &rcThis);
        }
    }

    // TRACE("(%d,%d)Rendering: %s %d\n", rcDraw.Width(), rcDraw.Height(),

```

```

//                                     *(GetDIB()->GetName()), m_wImOp &
OPACITY_MASK);

// Modify the source x and y values for the current phase of the
sprite.
// IMAGE_VERTICAL added Feb 9 mon '98
// I found these formula by the law of 'symmetry'!
// What a wonderful thing the symmetry is!!!
int ys = (m_wImOp & IMAGE_VERTICAL)
        ? (m_iCellRow + 1) * m_iCellHeight - (rcDraw.bottom - m_y)
        : m_iCellRow * m_iCellHeight + rcDraw.top - m_y;
ASSERT(ys >= 0 && ys < CSprite::GetHeight());
int xs = (m_wImOp & IMAGE_FLIP)
        ? (m_iCellColumn + 1) * m_iCellWidth - (rcDraw.right - m_x)
        : m_iCellColumn * m_iCellWidth + rcDraw.left - m_x;
ASSERT(xs >= 0 && xs < CSprite::GetWidth());
GetDIB()->CopyBits(pDIB,                // Destination DIB
rcDraw.left, rcDraw.top,                // Destination x, y
rcDraw.right - rcDraw.left,            // Width
rcDraw.bottom - rcDraw.top,            // Height
xs, ys,                                // Source x, y
PALETTEINDEX(m_btTransIndex),
m_wImOp); // Transparent color index
}

// Added by Kim, Soomin, Dec3 tue'96, to use for frame animation as in Login
Screen
// Draw the DIB to a given DC.
void CPhasedSprite::Draw(CDC* pDC, const CPoint& point)
{
    int ys = CSprite::GetHeight() - (m_iCellRow + 1) * m_iCellHeight; //
bottom-up aligned DIB
    ASSERT(ys >= 0 && ys < CSprite::GetHeight());
    int xs = m_iCellColumn * m_iCellWidth;
    ASSERT(xs >= 0 && xs < CSprite::GetWidth());
    ::StretchDIBits(pDC->GetSafeHdc(),
point.x, point.y, GetWidth(), GetHeight(), //
Destination x, y, width, height
xs, ys, GetWidth(), GetHeight(), // Source x, y, width,
height
GetDIB()->GetBitsAddress(), // Pointer to
bits
GetDIB()->GetBitmapInfoAddress(), // BITMAPINFO
DIB_RGB_COLORS, // Options
SRCCOPY); // Raster
operation code (ROP)
}

////////////////////////////////////
// Animation

void CPhasedSprite::SetAlarmTick(const DWORD dwAlarm)
{
    m_dwAlarmTick = dwAlarm ? dwAlarm : ::GetTickCount();
}

//=====

```

```

// OnIdle
// We use m_bi for animation type and m_ci for animation indices.
BOOL CPhasedSprite::HeartBeat(const DWORD dwCurTick)
{
    const static wIOSeq[] =
    {
        OPACITY_12, OPACITY_25, OPACITY_50, OPACITY_75, OPACITY_100,
        OPACITY_75, OPACITY_50, OPACITY_25, OPACITY_12, OPACITY_0
    };
    switch (GetAniType())
    {
    case SPRITE_ANI_REPEAT:
        SetNextCell();
        SetAlarmTick(dwCurTick + GetMSPT());
        break;
    case SPRITE_ANI_FADE:
        {
            WORD wIO = GetImOp();
            SetImOp((wIO & ~OPACITY_MASK) | wIOSeq[m_ai++]);
            if (m_ai >= sizeof(wIOSeq)/sizeof(wIOSeq[0]))
            {
                m_ai = 0;
                SetNextCell();
            }
            // Set Alarm time to be called for the next frame
            if (m_ai == 5) // OPACITY_100
                SetAlarmTick(dwCurTick + GetMSPT()*20);
            else
                SetAlarmTick(dwCurTick + GetMSPT());
            break;
        }
    case SPRITE_ANI_RANDOM:
        if ((m_iCellRow == 0) && (m_iCellColumn == 0))
            SetAlarmTick(dwCurTick + GetMSPT()*(1 + rand() % 100));
        else
            SetAlarmTick(dwCurTick + GetMSPT());
        SetNextCell();
        break;
    default:
        return FALSE;
    }
    return TRUE;
}

int CPhasedSprite::GetLinkType() const
{
    if (m_strHLink.GetLength() < 3)
        return HLINK_EMPTY;
    switch (m_strHLink[0])
    {
    case 'e':    if (m_strHLink[1] == ':')    return HLINK_U2_ENTRY; // e:
    case 'x':    if (m_strHLink[1] == ':')    return HLINK_U2_EXIT;  // x:
    case 'h':
    case 'f':    return HLINK_HTTP;           // http://, ftp://
    }
    return HLINK_EMPTY;
}
/*

```

```

BOOL CPhasedSprite::ParseHyperlink(CString& strHL, const int n)
{
    int nID = n;
    strHL = m_strHLink;

    while (nID--)
    {
        int i = strHL.Find('|');
        if (i >= 0)
            strHL = strHL.Left(i);
    }
}
*/

```

```

//
//  CPhasedSprite:
//
//  (C) Programmed by Kim,
//
//  Information Technology Institute
//  UNICHAT INC
//
#ifndef __PHSPRITE_H
#define __PHSPRITE_H

// index counts from 0
// i = r * Nc + c
// r = i / Nc
// c = i % Nc

#include "Sprite.h"

class CDIB;

class AFX_EXT_CLASS CPhasedSprite : public CSprite
{
    DECLARE_SERIAL(CPhasedSprite)
public:
    CPhasedSprite();
    virtual ~CPhasedSprite();

    CPhasedSprite& operator=(const CPhasedSprite& rhs);
    BOOL operator==(const CPhasedSprite& rhs);

    enum HYPERLINK_TYPE
    {
        HLINK_EMPTY,
        HLINK_U2_ENTRY,
        HLINK_U2_EXIT,
        HLINK_HTTP
    };

    // Attributes
    int GetNumCells() const { return
m_iNumCellRows*m_iNumCellColumns; }
    int GetNumCellRows() const { return m_iNumCellRows; }
    int GetNumCellColumns() const { return m_iNumCellColumns; }
    int GetCellRow() const { return m_iCellRow; }
    int GetCellColumn() const { return m_iCellColumn; }
    int GetCellID() const
    { return (m_iCellRow*m_iNumCellColumns + m_iCellColumn); }
    // from base classes
    virtual int GetHeight() const { return m_iCellHeight; }
    virtual int GetWidth() const { return m_iCellWidth; }
    BOOL HasHyperlink() const { return (!m_strHLink.IsEmpty()); }
    int GetLinkType() const;
    CString* GetHyperlink() { return &m_strHLink; }
    // BOOL ParseHyperlink(CString& strHL, const int n);

    // Operations
    virtual void Serialize(CArchive& ar);

```



```

    BOOL SetNumCells(const int iNumRows, const int iNumColumns);
    BOOL SetNumCells(const CSize s) { return SetNumCells(s.cy, s.cx); }
    BOOL SetNumCellRows(const int iNumRows);
    BOOL SetNumCellColumns(const int iNumColumns);
    BOOL SetCell(const int nID)
        { return SetCell(nID / m_iNumCellColumns, nID %
m_iNumCellColumns); }
    BOOL SetCell(const int iRow, const int iColumn);
    BOOL SetCell(const CPoint pt) { return SetCell(pt.y, pt.x); }
    BOOL SetCellRow(const int iRow);
    BOOL SetCellColumn(const int iColumn);
    BOOL SetNextCell();
    virtual void Draw(CDC* pDC, const CPoint& point);

    // from base classes
    virtual BOOL HitTest(const CPoint& point) const;
    virtual void Render(CDIB* pDIB, const CRect* pClipRect=NULL);
    virtual void Initialize();
    void SetHyperlink(const CString& strHL) { m_strHLink =
strHL; }

    // for Animation
    DWORD GetAlarmTick() const { return m_dwAlarmTick; }
    int GetMSPT() const { return m_nMSPT; }
}

    void SetMSPT(const int nMSPT)
        { m_nMSPT = nMSPT; SetAlarmTick(); }
    void SetAlarmTick(const DWORD dwAlarm=0L);
    virtual BOOL HeartBeat(const DWORD dwCurTick); // ticker

protected:
    int m_iNumCellRows; // Number of rows in the image grid
    int m_iNumCellColumns; // Number of columns in the image grid
    int m_iCellRow; // Current cell row
    int m_iCellColumn; // Current cell column
    int m_iCellHeight; // Height of a row
    int m_iCellWidth; // Width of a column
    // for Animation
    int m_ai; // current animation index
    int m_nMSPT; // Milliseconds per Tick
    DWORD m_dwAlarmTick; // to set alarm tick to be entered next
    CString m_strHLink; // Hyperlink
    "http://www.unichat.com/|http://..."
};
#endif // __PHSPRITE_H

```

```

//
//  CPSButton
//
//  (C) Programmed by Kim
//
//  Information Technology Institue
//  UNICHAT INC
//

#include "stdafx.h"
#include "PSButton.h"
#include "PhSprite.h"
#include "resource.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CPSButton

CPSButton::CPSButton()
{
    Init();
}

CPSButton::~CPSButton()
{
    if (m_pPS)
        delete m_pPS;
}

void CPSButton::Init()
{
    m_pPS = NULL;
    m_pPal = NULL;
}

BOOL CPSButton::Load(const CString& filename, const int nRows)
{
    // TRACE("%s\n", filename);
    if (m_pPS)
        delete m_pPS;
    m_pPS = new CPhasedSprite;
    if (!m_pPS->Load(filename))
    {
        delete m_pPS;
        m_pPS = NULL;
        AfxMessageBox("Failed to load Button DIB file");
        return FALSE;
    }
    m_pPS->SetNumCells(nRows, 1);
    return TRUE;
}

```

```

BOOL CPSButton::Load(const WORD wResid, const int nRows)
{
    if (m_pPS)
        delete m_pPS;
    m_pPS = new CPhasedSprite;
    if (!m_pPS->Load(wResid))
    {
        delete m_pPS;
        m_pPS = NULL;
        AfxMessageBox("Failed to load Button DIB file");
        return FALSE;
    }
    m_pPS->SetNumCells(nRows, 1);
    return TRUE;
}

void CPSButton::Resize()
{
    if (m_pPS)
        SetWindowPos(NULL, -1, -1, m_pPS->GetWidth(), m_pPS->GetHeight(),
                      SWP_NOMOVE | SWP_NOZORDER | SWP_NOREDRA
                      | SWP_NOACTIVATE);
}

void CPSButton::MoveResize(const int x, const int y)
{
    if (m_pPS)
        SetWindowPos(NULL, x, y, m_pPS->GetWidth(), m_pPS->GetHeight(),
                      SWP_NOZORDER | SWP_NOREDRA
                      | SWP_NOACTIVATE);
}

// virtual D:\MsDev\MFC\src\WINBTN.CPP(113)
void CPSButton::DrawItem(LPDRAWITEMSTRUCT lpDIS)
{
    ASSERT(lpDIS != NULL);
    if (!m_pPS)
        return;
    m_pPS->SetCell(PSBTN_NORMAL, 0);
    UINT state = lpDIS->itemState;
    if ((state & ODS_SELECTED) && (PSBTN_SELECTED < m_pPS-
    >GetNumCellRows()))
        m_pPS->SetCell(PSBTN_SELECTED, 0);
    else if (state & ODS_FOCUS)
    {
        if (PSBTN_FOCUS < m_pPS->GetNumCellRows())
            m_pPS->SetCell(PSBTN_FOCUS, 0);
        else
            m_pPS->SetCell(PSBTN_NORMAL, 0);
    }
    else if (state & ODS_DISABLED)
    {
        if (PSBTN_DISABLED < m_pPS->GetNumCellRows())
            m_pPS->SetCell(PSBTN_DISABLED, 0);
        else
            m_pPS->SetCell(PSBTN_NORMAL, 0);
    }
}

```

```

CDC* pDC = CDC::FromHandle(lpDIS->hDC);
CPalette* pPalOld = NULL;
if (m_pPal)
    pPalOld = pDC->SelectPalette(m_pPal, FALSE);

CRect rect;
rect.CopyRect(&lpDIS->rcItem);
if (m_pPS)
    m_pPS->Draw(pDC, rect.TopLeft());
if (pPalOld)
    pDC->SelectPalette(pPalOld, FALSE);
}

BEGIN_MESSAGE_MAP(CPSButton, CButton)
    //{{AFX_MSG_MAP(CPSButton)
    ON_WM_SETCURSOR()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CPSButton message handlers

BOOL CPSButton::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    // if (!(GetState() & 0x0008)) // check if it doesn't have a focus yet
    //     SetState(TRUE); // then highlights it.
    // commented out due to flickering problems on Rendering
    // ::SetCursor(AfxGetApp()->LoadCursor(IDC_HARROW));
    return TRUE;
    // return CButton::OnSetCursor(pWnd, nHitTest, message);
}

```

```

//
//  CPSButton : 256 palette-color button
//
//  (C) Programmed by Kim
//
//  Information Technology Institue
//  UNICHAT INC
//
#ifndef __PSBUTTON_H
#define __PSBUTTON_H

////////////////////////////////////
// CPSButton window

#include "PhSprite.h"

// PhasedSprite Button to draw with 256-color palette
// Button DIB image should align cells only in rows
class AFX_EXT_CLASS CPSButton : public CButton
{
public:
    CPSButton();
    void Init();
    void SetPalette(CPalette* pPal) { m_pPal = pPal; } // follow
the background's palette
    BOOL Load(const CString& filename, const int nRows=3);
    BOOL Load(const WORD wResid, const int nRows=3);
    void Resize(); // call after SubclassDlgItem in OnInitialDialog of
the parent
    void MoveResize(const int x, const int y);
    void MoveResize(const CPoint& ptLT) { MoveResize(ptLT.x, ptLT.y); }

    int GetWidth() const { return m_pPS->GetWidth(); }
    int GetHeight() const { return m_pPS->GetHeight(); }
    CDIB* GetDIB() { return m_pPS ? m_pPS->GetDIB() : NULL; }
}

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CPSButton)
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CPSButton();

    // Generated message map functions
protected:
    enum
    {
        PSBTN_NORMAL=0, PSBTN_SELECTED, PSBTN_FOCUS, PSBTN_DISABLED
    } PSBUTTON_STATE;
    virtual void DrawItem(LPDRAWITEMSTRUCT lpDIS);

    CPhasedSprite* m_pPS;
    CPalette* m_pPal;

```

UC2Ani\PSButton.h

```
//{{AFX_MSG(CPSButton)
afx_msg BOOL OnSetCursor(CWnd* pwnd, UINT nHitTest, UINT message);
//}}AFX_MSG
```

```
DECLARE_MESSAGE_MAP()
```

```
};
```

```
////////////////////////////////////
#endif // __PSBUTTON_H
```

=====

MICROSOFT FOUNDATION CLASS LIBRARY : UC2Ani

=====

AppWizard has created this UC2Ani DLL for you. This DLL not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your DLL.

This file contains a summary of what you will find in each of the files that make up your UC2Ani DLL.

UC2Ani.cpp

This is the main DLL source file that contains the definition of DllMain().

UC2Ani.rc

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

res\UC2Ani.rc2

This file contains resources that are not edited by Microsoft Developer Studio. You should place all resources not editable by the resource editor in this file.

UC2Ani.def

This file contains information about the DLL that must be provided to run with Microsoft Windows. It defines parameters such as the name and description of the DLL. It also exports functions from the DLL.

UC2Ani.clw

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

////////////////////////////////////
Other standard files:

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header (PCH) file named UC2Ani.pch and a precompiled types file named StdAfx.obj.

Resource.h

This is the standard header file, which defines new resource IDs. Microsoft Developer Studio reads and updates this file.

////////////////////////////////////
Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you should add to or customize.

////////////////////////////////////

UC2Ani\ReadMe.txt


```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by UC2Ani_Kor.rc
//
#define IDC_HARROW                                139

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE                130
#define _APS_NEXT_COMMAND_VALUE                32771
#define _APS_NEXT_CONTROL_VALUE                1000
#define _APS_NEXT_SYMED_VALUE                  101
#endif
#endif
```

```

//
//  CSound
//
//  (C) Programmed by Kim,
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "sound.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CSound

IMPLEMENT_SERIAL(CSound, CObject, 0 /* Schema number*/ )

CSound::CSound()
{
    m_pSoundImage = NULL;
}

CSound::~CSound()
{
    if (m_pSoundImage)
        delete [] m_pSoundImage;
}

////////////////////////////////////
// CSound serialization

void CSound::Serialize(CArchive& ar)
{
    ASSERT(1); // Not supported
}

////////////////////////////////////
// CSound commands

BOOL CSound::Load(const char* pszFileName)
{
    CString strFile;

    if (lstrlen(pszFileName) == 0)
    {
        // Show an File Open dialog to get the name.
        CFileDialog dlg (TRUE, // Open
            NULL, // No default extension
            NULL, // No initial file name
            OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
            "Wave files (*.WAV)|*.WAV|All files
(*.*)|*.*)|");
    }

```

```

        if (dlg.DoModal() == IDOK)
            strFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        strFile = pszFileName;
    }

    // Try to open the file for read access.
    CFile file;
    if (!file.Open(strFile, CFile::modeRead | CFile::shareDenyWrite))
    {
        TRACE("%s not found\n", strFile);
        strFile += " not found!";
        AfxMessageBox(strFile);
        return FALSE;
    }

    BOOL bResult = Load(&file);
    file.Close();
    if (!bResult)
    {
        AfxMessageBox("Failed to load wave file");
        return bResult;
    }
}

BOOL CSound::Load(CFile* fp)
{
    // Load the whole file into memory.
    // Free any existing memory.
    if (m_pSoundImage)
        delete [] m_pSoundImage;
    // Allocate a new block big enough for the entire file.
    int iLength = fp->GetLength();
    m_pSoundImage = new BYTE[iLength];
    ASSERT(m_pSoundImage);
    // Read the entire file into memory.
    int iBytes = fp->Read(m_pSoundImage, iLength);
    ASSERT(iBytes == iLength);
    return TRUE;
}

BOOL CSound::Load(const WORD wResid)
{
    ASSERT(wResid);
    HINSTANCE hInst = AfxGetResourceHandle();
    HRSRC hrsrc = ::FindResource(hInst, MAKEINTRESOURCE(wResid), "WAVE");
    if (!hrsrc)
    {
        TRACE("WAVE resource not found");
        return FALSE;
    }
    HGLOBAL hg = LoadResource(hInst, hrsrc);
    if (!hg)
    {

```

```
        TRACE("Failed to load WAVE resource");
        return FALSE;
    }
    m_pSoundImage = (BYTE*)LockResource(hg);
    ASSERT(m_pSoundImage);
    return TRUE;
}

BOOL CSound::Play()
{
    if (!m_pSoundImage)
    {
        TRACE("Nothing to play");
        return FALSE;
    }
    return ::PlaySound((LPCSTR)m_pSoundImage, NULL,
                       SND_MEMORY | SND_ASYNC | SND_NODEFAULT);
}
```

```

//
//      CSound
//
//      (C) Programmed by Kim
//UNICHAT
//      Information Technology Institute
//      UNICHAT INC
//
#ifdef __SOUND__
#define __SOUND__

#include <mmsystem.h>

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CSound object

class AFX_EXT_CLASS CSound : public CObject
{
    DECLARE_SERIAL(CSound)
public:
    CSound();
    virtual ~CSound();
    BOOL Load(const char* pszFileName=NULL);
    BOOL Load(CFile* fp);
    BOOL Load(const WORD wResid);
    BOOL Play();

// Implementation
protected:
    // Overridden for document i/o
    virtual void Serialize(CArchive& ar);

private:
    BYTE* m_pSoundImage;
};

#endif // __SOUND__

```

```

//
//  CSpriteNotifyObj :
//
//  (C) Programmed by Kim
//  SDS Media Lab
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "dib.h"
#include "dibpal.h"
#include "spriteno.h"
#include "splstno.h"
#include "sprite.h"
#include "spritlst.h"
#include "osbview.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CSpriteListNotifyObj

CSpriteListNotifyObj::CSpriteListNotifyObj()
{
    m_pSpriteList = NULL;
    m_pBufferView = NULL;
}

CSpriteListNotifyObj::~CSpriteListNotifyObj()
{
}

// Notification callback from a CSprite object.
void CSpriteListNotifyObj::Change(CSprite* pSprite, CHANGETYPE change,
                                   CRect* pRect1, CRect* pRect2)
{
    if (change & CSpriteNotifyObj::ZORDER)
    {
        // Reposition the sprite in the z-order list.
        ASSERT(m_pSpriteList);
        m_pSpriteList->Reorder(pSprite);
        // Add the sprite position to the dirty list.
        ASSERT(m_pBufferView);
        m_pBufferView->AddDirtyRegion(pRect1);
    }
    if (change & CSpriteNotifyObj::POSITION)
    {
        // pRect1 and pRect2 point to old and new rectangle positions;
        // add these rectangles to the dirty list.
        ASSERT(m_pBufferView);
        m_pBufferView->AddDirtyRegion(pRect1);
        m_pBufferView->AddDirtyRegion(pRect2);
    }
}

```

```
if (change & CSpriteNotifyObj::IMAGE)
{
    // redraw the sprite
    // Add the sprite position to the dirty list.
    ASSERT(m_pBufferView);
    m_pBufferView->AddDirtyRegion(pRect1);
}
}
```

```

//
//  CSpriteListNotifyObj;
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
//  This is a class derived from CSpriteNotifyObj which is used in
//  CSpriteList to handle notification calls from CSprite objects.
//

#ifndef __SPLSTNO_H
#define __SPLSTNO_H

class CSpriteList;
class COSBView;

#include "SpriteNo.h"

class AFX_EXT_CLASS CSpriteListNotifyObj : public CSpriteNotifyObj
{
public:
    CSpriteListNotifyObj();
    ~CSpriteListNotifyObj();
    void SetList(CSpriteList* pSpriteList)    { m_pSpriteList =
pSpriteList; }
    void SetView(COSBView* pBufferView)      { m_pBufferView =
pBufferView; }
    COSBView* GetView()                      { return m_pBufferView; }
    void Change(CSprite* pSprite, CHANGETYPE change,
               CRect* pRect1=NULL, CRect* pRect2=NULL);

protected:
    CSpriteList*      m_pSpriteList;
    COSBView*        m_pBufferView;
};
#endif // __SPLSTNO_H

```



```

//
//  CSprite :
//
//  (C) Programmed by Kim, 1995-96
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "DIB.h"
#include "SpriteNo.h"
#include "Sprite.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CSprite

IMPLEMENT_SERIAL(CSprite, CObject, 0 /* Schema number */)

CSprite::CSprite()
{
    m_pDIB = NULL;
    m_bDeleteDIB = TRUE;
    m_wImOp = DEFAULT_IO;
    m_x = m_y = m_z = m_e = 0;
    m_sEarth = CSize(0, 0);
    m_wType = SPRITE_STATIC;
    m_btTransIndex = 0;
    m_pNotifyObj = NULL;
}

CSprite::~CSprite()
{
    if (m_bDeleteDIB && m_pDIB)
        delete m_pDIB;
}

CSprite& CSprite::operator=(CSprite& rhs)
{
    if (this == &rhs)
        return *this;
    SetZ(rhs.GetZ());
    if (rhs.IsResManUsed())
        SetDIB(rhs.GetDIB());
    else
    {
        ASSERT(rhs.GetDIB());
        if (m_pDIB)
            delete m_pDIB;
        m_pDIB = new CDIB;
        *m_pDIB = *rhs.GetDIB();
        Initialize();
    }
}

```

```

    }
    SetLT(rhs.GetLT());
    SetImOp(rhs.GetImOp());
    SetElevation(rhs.GetElevation());
    SetType(rhs.GetType());
    m_pNotifyObj = rhs.m_pNotifyObj;
    return *this;
}

BOOL CSprite::operator==(CSprite& rhs)
{
    return ((GetZ() == rhs.GetZ()) &&
            (GetDIB() == rhs.GetDIB()) &&
            (IsResManUsed() == rhs.IsResManUsed()) &&
            (GetLT() == rhs.GetLT()) &&
            (GetImOp() == rhs.GetImOp()) &&
            (GetElevation() == rhs.GetElevation()));
}

// Set the initial state of the sprite from its DIB image.
void CSprite::Initialize()
{
    // Get the address of the top-left pixel.
    BYTE* p = (BYTE*)GetDIB()->GetPixelAddress(0, 0);
    ASSERT(p);
    // Get the pixel value and save it.
    m_btTransIndex = *p;
}

void CSprite::SetDIB(CDIB* pDIB)
{
    m_pDIB = pDIB;
    m_bDeleteDIB = FALSE;
    Initialize();
}

// Set a new left-top x, y position.
// This will move m_rcBase by the displacements
// Notify to update the old and new rectangle area
void CSprite::SetLT(const int x, const int y)
{
    if (x == m_x && y == m_y)
        return;
    // Save the current position.
    CRect rcOld;
    GetRect(rcOld);
    // Move to new position.
    m_x = x;
    m_y = y;
    CRect rcNew;
    GetRect(rcNew);
    // Notify that we have moved from our old position to our new position.
    if (m_pNotifyObj) // for sprites inserted into CSpriteList
        m_pNotifyObj->Change(this, CSpriteNotifyObj::POSITION, &rcOld,
&rcNew);
}

```

```

////////////////////////////////////
// CSprite serialization

```

```

void CSprite::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar << (DWORD)m_x;
        ar << (DWORD)m_y;
        ar << (DWORD)m_z;
    }
    else
    {
        DWORD dw;
        ar >> dw; m_x = (int)dw;
        ar >> dw; m_y = (int)dw;
        ar >> dw; m_z = (int)dw;
        // Now generate the other parameters from the DIB.
        Initialize();
    }
}

```

```

////////////////////////////////////
// CSprite commands

```

```

// Render a sprite to a DIB.
void CSprite::Render(CDIB* pDIB, const CRect* pClipRect)
{
    ASSERT(pDIB);
    // Get the sprite rectangle.
    CRect rcThis;    // This sprite's region
    GetRect(rcThis);
    CRect rcDraw;
    // See if the sprite is visible inside the rectangle.
    if (pClipRect)
    {
        if (!rcDraw.IntersectRect(pClipRect, &rcThis))
            return; // not visible
    }
    else
    {
        // For images with opacity,
        // set drawing area to contain the whole sprite rectangle.
        // So, withdraw pClipRect. pClipRect is just for hit test.
        // This is to prevent "different initial position problem" Feb 21 '98
        if (!pClipRect || (m_wImOp & OPACITY_MASK))
        {
            CRect rcOSB;
            pDIB->GetRect(rcOSB);
            rcDraw.IntersectRect(&rcOSB, &rcThis);
        }

        // Copy the image of the sprite.
        GetDIB()->CopyBits(pDIB,
            rcDraw.left, rcDraw.top,
            rcDraw.right - rcDraw.left,
            rcDraw.bottom - rcDraw.top,
            rcDraw.left - m_x,
            // Dest DIB
            // Dest x, y
            // Width
            // Height
            // Source x

```

```

        rcDraw.top - m_y,                // Source y
        PALETTEINDEX(m_btTransIndex),
        m_wImOp);                        // Transparent color index, Image Operation
    }

// Load a sprite image from a disk file.
BOOL CSprite::Load(const char* pszFileName, const char* pszPalFileName)
{
    if (!m_pDIB)
        m_pDIB = new CDIB;
    if (!GetDIB()->Load(pszFileName, pszPalFileName))
    {
        delete m_pDIB;
        m_pDIB = NULL;
        return FALSE;
    }
    // Make sure this is an 8 bpp DIB
    if (GetDIB()->GetBitCount() != 8)
    {
        AfxMessageBox("Only 8 bpp DIBs are supported");
        delete m_pDIB;
        m_pDIB = NULL;
        return FALSE;
    }

    Initialize();
    return TRUE;
}

// Load a sprite image from a disk file.
BOOL CSprite::Load(CFile* const fp, const char* pszPalFileName)
{
    if (!m_pDIB)
        m_pDIB = new CDIB;
    if (!GetDIB()->Load(fp, pszPalFileName))
    {
        delete m_pDIB;
        m_pDIB = NULL;
        return FALSE;
    }
    Initialize();
    return TRUE;
}

// Load a sprite image from a resource.
BOOL CSprite::Load(const WORD wResid)
{
    if (!m_pDIB)
        m_pDIB = new CDIB;
    if (!GetDIB()->Load(wResid))
    {
        delete m_pDIB;
        m_pDIB = NULL;
        return FALSE;
    }
    Initialize();
    return TRUE;
}

```

```

}

// Map colors to palette.
BOOL CSprite::MapColorsToPalette(CPalette* pPal)
{
    BOOL bResult = GetDIB()->MapColorsToPalette(pPal);
    // Get the transparency info again.
    // Note: Local call only; don't call any derived class.
    CSprite::Initialize();
    return bResult;
}

// Get the bounding rectangle.
void CSprite::GetRect(CRect& rect) const
{
    rect.left    = m_x;
    rect.top     = m_y;
    rect.right   = m_x + GetWidth();
    rect.bottom  = m_y + GetHeight();
}

// Test for a hit in a non-transparent area
BOOL CSprite::HitTest(const CPoint& point) const
{
    // Test if the point is inside the sprite rectangle
    if ((point.x > m_x) && (point.x < m_x + GetWidth())
        && (point.y > m_y) && (point.y < m_y + GetHeight()))
    {
        // See if this point is transparent by testing whether the pixel
        value // is the same as the top left corner value.
        // Note that top left of the image is bottom left in the DIB.
        BYTE* p = (BYTE*)m_pDIB->GetPixelAddress(point.x - m_x, point.y -
m_y);
        if (*p != m_btTransIndex)
            return TRUE; // hit
        }
    return FALSE;
}

// Set a new Z-order.
void CSprite::SetZ(const int z)
{
    if (m_z != z)
    {
        m_z = z;
        // See if we have to notify anyone.
        if (m_pNotifyObj)
        {
            CRect rc;
            GetRect(rc);
            m_pNotifyObj->Change(this, CSpriteNotifyObj::ZORDER, &rc);
        }
    }
}

/*

```

```

int CSprite::SetZByGroup(const int nG)
{
    int nZO = -nG*GetGroupHeight() - GetEarthPointY();
    // TRACE("Z Order=%d\n", nZO);
    SetZ(nZO);
    return nZO;
}
*/
// virtual
void CSprite::SetImOp(const WORD wImOp)
{
    if (m_wImOp != wImOp)
    {
        m_wImOp = wImOp;
        // See if we have to notify anyone.
        if (m_pNotifyObj)
        {
            CRect rc;
            GetRect(rc);
            m_pNotifyObj->Change(this, CSpriteNotifyObj::IMAGE, &rc);
        }
    }
}

BOOL CSprite::IsSameDIB(CSprite* pS) const
{
    if (!m_pDIB || !pS)
        return FALSE;
    if (IsResManUsed())
        return (m_pDIB->GetName() == pS->GetDIB()->GetName());
    CString* pS1 = m_pDIB->GetName();
    CString* pS2 = pS->GetDIB()->GetName();
    return (*pS1 == *pS2); // Compare the string value
}

// The y pixel coord. of the ground (earth)
int CSprite::GetEarthPointY() const
{
    if (GetSrcType() == SPRITE_TILE)
    {
        if (m_e > 0) // For an elevated tile, look it as a normal
            return (m_y + GetHeight()/2 - 1 + m_e);
        // for a tile with elev<=0, give it somewhat high priority in
        return (m_y - GetHeight()*4 + m_e);
    }
    return (m_y + m_sEarth.cy + m_e);
}

```

UC2Ani\Sprite.h

```
//
//  CSprite:
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#ifndef __SPRITE_H
#define __SPRITE_H

class CSpriteNotifyObj;

#include "DIB.h"

enum SPRITE_TYPE
{
    SPRITE_TILE           = 0x0001,
    SPRITE_WALL           = 0x0002,
    SPRITE_STATIC         = 0x0004,
    SPRITE_PHASED        = 0x0008,
    SPRITE_ACTOR          = 0x0010,
    SPRITE_SRC_MASK       = 0x00FF,    // lower 1 byte
    SPRITE_ANI_REPEAT     = 0x0100,
    SPRITE_ANI_FADE       = 0x0200,
    SPRITE_ANI_RANDOM     = 0x0400,
    SPRITE_ANI_ACTOR      = 0x1000,
    SPRITE_ANI_MASK       = 0xFF00
};

class AFX_EXT_CLASS CSprite : public CObject
{
    DECLARE_SERIAL(CSprite)
public:
    CSprite();
    virtual ~CSprite();
    CSprite& operator=(CSprite& rhs);
    BOOL operator==(CSprite& rhs);

// Attributes
    CDIB* GetDIB()      { return m_pDIB; }
    virtual int         GetWidth() const { return m_pDIB->GetWidth(); }
    // Image width
    virtual int         GetHeight() const { return m_pDIB->GetHeight(); }
    // Image height

    BOOL IsImageFlip()   const          { return m_wImOp & IMAGE_FLIP; }
    BOOL IsImageVertical() const { return m_wImOp & IMAGE_VERTICAL; }
    BOOL IsSameDIB(CSprite* pS) const;
    WORD GetImOp() const { return m_wImOp; }
    WORD GetType() const { return m_wType; }
    WORD GetSrcType() const { return (m_wType & SPRITE_SRC_MASK); }
    WORD GetAniType() const { return (m_wType & SPRITE_ANI_MASK); }
    BOOL IsResManUsed() const { return (!m_bDeleteDIB); }
    CString* GetName() { return (m_pDIB ? m_pDIB->GetName() : NULL); }
};
```

```

    int      GetX() const { return m_x; } // Get x.
    int      GetY() const { return m_y; } // Get y.
    int      GetZ() const { return m_z; } // Get z-order.
    int      GetElevation() const { return m_e; }
    CPoint    GetLT() const { return CPoint(m_x, m_y); }
    CPoint    GetCenter() const { return CPoint(m_x + GetWidth()/2, m_y +
GetHeight()/2); }
    virtual void GetRect(CRect& rect) const;
    int      GetEarthPointY() const;
    CPoint    GetEarthPoint() const
        { return CPoint(m_x + m_sEarth.cx, GetEarthPointY()); }
    CSize GetEarth() const { return m_sEarth; }
    int      GetEarthCY() const { return m_sEarth.cy; }
//    int      GetZGroup() const { return (GetZ()/GetGroupHeight()); }

// Operations
    virtual void Serialize(CArchive& ar);
    void SetDIB(CDIB* pDIB);
    virtual BOOL Load(CFile* const fp, const char* pszPalFileName=NULL);
        // Load from file
    virtual BOOL Load(const char* pszFileName=NULL, const char*
pszPalFileName=NULL); // Load DIB from disk file
    virtual BOOL Load(const WORD wResid); // Load DIB
from resource
    void SetNotificationObject(CSpriteNotifyObj* pNO) { m_pNotifyObj =
pNO; }

// Bitmap
    virtual BOOL MapColorsToPalette(CPalette* pPal);
    virtual void Render(CDIB* pDIB, const CRect* pClipRect=NULL);
    virtual void SetImOp(const WORD wImOp);
    void SetOpacity(const WORD wOpacity)
        { SetImOp((m_wImOp & ~OPACITY_MASK) | wOpacity); }
    void SetOrient(const WORD wOrient) // Mirror Stage
        { SetImOp((m_wImOp & ~ORIENT_MASK) | wOrient); }

    void SetType(const WORD wSrcType, const WORD wAniType=0)
        { m_wType = (wSrcType | wAniType); }
    void SetSrcType(const WORD wSrcType)
        { m_wType = (wSrcType | (m_wType & ~SPRITE_SRC_MASK)); }
    void SetAniType(const WORD wAniType)
        { m_wType = ((m_wType & ~SPRITE_ANI_MASK) | wAniType); }

// for Actors
    int      TickAll(); // CPXClientApp::OnIdle calls this
function

// Coordinates
    virtual BOOL HitTest(const CPoint& point) const;
    virtual void SetLT(const int x, const int y);
    void SetLT(const CPoint& point) { SetLT(point.x, point.y); }
    void SetZ(const int z);
    void Unelevate() { m_y += m_e; } // Temporal use for rendering
    void Elevate() { m_y -= m_e; } // "
//    int      SetZByGroup(const int nG);
    void SetZByEarth() { SetZ(-GetEarthPointY()); }
    void SetEarth(const CSize sEarth) { m_sEarth = sEarth; }

```



```

void SetElevation(const int e)          { m_e = e; }
void IncElevation(const int ie)         { m_e += ie; }
void MoveBy(const int xs, const int ys) { SetLT(m_x + xs, m_y + ys); }
}

void MoveToEarth(CPoint& point)
    { SetLT(point.x - m_sEarth.cx, point.y - m_sEarth.cy -
m_e); }
void MoveToCenter(CPoint& point)
    { SetLT(point.x - GetWidth()/2, point.y - GetHeight()/2); }

protected:
    virtual void Initialize();
//    int GetGroupHeight() const { return 1000; } // >
Screen.cy

CDIB* m_pDIB;
BYTE m_btTransIndex; // Transparency index value
WORD m_wImOp; // Bitmap operation options
BOOL m_bDeleteDIB; // Delete DIB on exit?
int m_x; // X Coordinate of top-left corner
int m_y; // Y Coordinate of top-left corner
int m_e; // Elevation
int m_z; // Z-order for sprite
CSize m_sEarth; // Center of Earth Point (offset)
WORD m_wType; // Sprite Type
CSpriteNotifyObj* m_pNotifyObj; // Pointer to a notification object
// CSize m_offset; // offset of the m_rcBase
// CRect m_rcBase; // for hit-test between actors
};
#endif // __SPRITE_H

```

```

//
//  CSpriteNotifyObj :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
//  spriteno.h : header file
//
//  CSpriteNotifyObj class: Nigel Thompson
//
//  This is a class of pure virtual functions with no data. It is used
//  by sprite objects to make notification callbacks. A user of the CSprite
//  class can derive an object from CSpriteNotifyObj and pass a pointer to
//  this
//  derived class object to the sprite object for notification calls.
//  Just like OLE's IClientSite interface really.
//

#ifndef __SPRITENO_H
#define __SPRITENO_H

class CSprite;

class AFX_EXT_CLASS CSpriteNotifyObj : public CObject
{
public:
    enum CHANGETYPE {
        ZORDER      = 0x0001,
        POSITION      = 0x0002,
        IMAGE         = 0x0004
    };

public:
    virtual void Change(CSprite* pSprite, CHANGETYPE change,
        CRect* pRect1=NULL, CRect* pRect2=NULL) =
        0;
};

#endif // __SPRITENO_H

```

```

//
//  CSpriteList :
//
//  (C) Programmed by Kim
//  UNICHAT
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "dib.h"
#include "spriteno.h"
#include "sprite.h"
#include "splstno.h"
#include "spritlst.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CSpriteList

IMPLEMENT_SERIAL(CSpriteList, COBList, 0 /* Schema number */ )

CSpriteList::CSpriteList()
{
    // Give the sprite notification object
    // a pointer to the list object.
    m_NotifyObj.SetList(this);
}

CSpriteList::~CSpriteList()
{
}

////////////////////////////////////
// CSpriteList serialization

void CSpriteList::Serialize(CArchive& ar)
{
    // Let the base class create the set of objects.
    COBList::Serialize(ar);

    // If we've just loaded, initialize each sprite.
    if (ar.IsLoading() && (m_NotifyObj.GetView()))
    {
        POSITION pos = GetHeadPosition();
        while (pos)
        {
            CSprite* pSprite = GetNext(pos); // Increment position.
            pSprite->SetNotificationObject(&m_NotifyObj);
        }
    }
}

```

```

////////////////////////////////////
// CSpriteList commands

```

```

// Add a sprite to the list, placing it according to its z-order value.

```

```

BOOL CSpriteList::Insert(CSprite* pNewSprite)

```

```

{
    // Set the notification object pointer in the sprite
    // to the sprite list's notification object.
    if (m_NotifyObj.GetView() // has associated view
        pNewSprite->SetNotificationObject(&m_NotifyObj);

    // Walk down the list until we either get to the end
    // or we find a sprite with the same or higher z-order
    // in which case we insert just before that one.

    POSITION pos = GetHeadPosition();
    POSITION posThis;
    CSprite* pSprite;
    while (pos)
    {
        posThis = pos;
        pSprite = GetNext(pos); // Increment position.
        if (pSprite->GetZ() >= pNewSprite->GetZ())
        {
            InsertBefore(posThis, pNewSprite);
            return TRUE;
        }
    }
    // Nothing with the same or a higher z-order, so add the sprite to
    // the end.
    AddTail(pNewSprite);
    return TRUE;
}

```

```

// Remove a sprite from the list, but do not delete it

```

```

CSprite* CSpriteList::Remove(CSprite* pSprite)

```

```

{
    POSITION pos = Find(pSprite);
    if (pos == NULL)
        return NULL;
    RemoveAt(pos);
    return pSprite;
}

```

```

// Remove everything from the list deleting all the sprites we remove

```

```

void CSpriteList::RemoveAll(const BOOL bDelete)

```

```

{
    if (bDelete)
    {
        // Walk down the list deleting objects as we go.
        // We need to do this here because the base class simply deletes
the pointers.
        POSITION pos = GetHeadPosition();
        CSprite* pSprite;
        while (pos)
        {
            pSprite = GetNext(pos); // Increment position.

```

```

        if (pSprite)
        {
            ASSERT(pSprite->IsKindOf(RUNTIME_CLASS(CSprite)));
            if (pSprite->GetSrcType() != SPRITE_TILE)
                delete pSprite;
        }
    }
    // Now call the base class to remove the pointers.
    COBList::RemoveAll();
}

// Move a sprite to its correct z-order position.
void CSpriteList::Reorder(CSprite* pSprite)
{
    // Remove the sprite from the list.
    if (!Remove(pSprite))
    {
        TRACE("Unable to find sprite");
        return; // Not there.
    }
    // Now insert it again in the right place.
    Insert(pSprite);
}

// Test for a mouse hit on any sprite in the list.
CSprite* CSpriteList::HitTest(const CPoint& point)
{
    // Walk the list top down.
    POSITION pos = GetHeadPosition();
    CSprite* pSprite;
    while (pos)
    {
        pSprite = GetNext(pos); // Increment position.
        if (pSprite->HitTest(point))
            return pSprite;
    }
    return NULL;
}

void CSpriteList::RemoveRedundantSprite(CSprite* pSprite)
{
    POSITION pos = GetTailPosition();
    POSITION posOld = pos;
    CSprite* pS;
    while (pos)
    {
        pS = (CSprite*)GetPrev(pos);
        if ((pS->GetZ() >= pSprite->GetZ()) &&
            (pS->GetSrcType() != SPRITE_TILE) &&
            (pS != pSprite) && // skip for tile or itself
            (*pS == *pSprite))
        {
            TRACE("RemoveRedundantSprite - (%d,%d)\n", pS->GetX(), pS-
>GetY());
            pS = (CSprite*)GetAt(posOld);
            RemoveAt(posOld);
        }
    }
}

```

```
        delete pS;  
    }  
    posOld = pos;  
}  
}
```

```

//
//  CSpriteList :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#ifndef __SPRITELIST__
#define __SPRITELIST__

#include "SpLstNo.h"

class AFX_EXT_CLASS CSpriteList : private CObList
{
    DECLARE_SERIAL(CSpriteList)
public:
    CSpriteList();
    virtual ~CSpriteList();
    int          GetCount() const { return CObList::GetCount(); }
    BOOL         Insert(CSprite* pSprite);
    void         Reorder(CSprite* pSprite);
    CSprite*     Remove(CSprite* pSprite);
    void         RemoveAll(const BOOL bDelete=TRUE);
    CSprite*     GetNext(POSITION &pos) { return
(CSprite*)CObList::GetNext(pos); }
    CSprite*     GetPrev(POSITION &pos) { return
(CSprite*)CObList::GetPrev(pos); }
    POSITION      GetTailPosition() const { return
CObList::GetTailPosition(); }
    POSITION      GetHeadPosition() const { return
CObList::GetHeadPosition(); }
    CSprite*     HitTest(const CPoint& point);
    virtual void Serialize(CArchive& ar);
    BOOL         IsEmpty() { return CObList::IsEmpty(); }
    void         RemoveRedundantSprite(CSprite* pSprite);

public:
    CSpriteListNotifyObj m_NotifyObj;
};
#endif // __SPRITELIST__

```

UC2Ani\StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//      UC2Ani.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```



```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H__C193E4D9_88DD_11D1_ACEA_080009B9F339__INCLUDED_)
#define AFX_STDAFX_H__C193E4D9_88DD_11D1_ACEA_080009B9F339__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>           // MFC core and standard components
#include <afxext.h>           // MFC extensions

#ifndef _AFX_NO_OLE_SUPPORT
#include <afxole.h>           // MFC OLE classes
#include <afxodlgs.h>         // MFC OLE dialog classes
#include <afxdisp.h>          // MFC OLE automation classes
#endif // _AFX_NO_OLE_SUPPORT

#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>             // MFC ODBC database classes
#endif // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>           // MFC DAO database classes
#endif // _AFX_NO_DAO_SUPPORT

#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>           // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

// UC2Ani
#include <mmsystem.h>

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__C193E4D9_88DD_11D1_ACEA_080009B9F339__INCLUDED_)

```

```

// UC2Ani.cpp : Defines the initialization routines for the DLL.
//
//      (C) Programmed by Kim
//
//      Information Technology Institute
//      UNICHAT INC
#include "stdafx.h"
#include <afxdll.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

static AFX_EXTENSION_MODULE UC2AniDLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    // Remove this if you use lpReserved
    UNREFERENCED_PARAMETER(lpReserved);

    if (dwReason == DLL_PROCESS_ATTACH)
    {
        TRACE0("UC2ANI.DLL Initializing!\n");

        // Extension DLL one-time initialization
        if (!AfxInitExtensionModule(UC2AniDLL, hInstance))
            return 0;

        // Insert this DLL into the resource chain
        // NOTE: If this Extension DLL is being implicitly linked to by
        // an MFC Regular DLL (such as an ActiveX Control)
        // instead of an MFC application, then you will want to
        // remove this line from DllMain and put it in a separate
        // function exported from this Extension DLL. The Regular DLL
        // that uses this Extension DLL should then explicitly call that
        // function to initialize this Extension DLL. Otherwise,
        // the CDynLinkLibrary object will not be attached to the
        // Regular DLL's resource chain, and serious problems will
        // result.

        new CDynLinkLibrary(UC2AniDLL);
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        TRACE0("UC2ANI.DLL Terminating!\n");
        // Terminate the library before destructors are called
        AfxTermExtensionModule(UC2AniDLL);
    }
    return 1;    // ok
}

```

```

//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif // _WIN32

#ifdef APSTUDIO_INVOKED
////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include \"afxres.h\"\\r\\n"
    "\\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\\r\\n"
    "#define _AFX_NO_OLE_RESOURCES\\r\\n"
    "#define _AFX_NO_TRACKER_RESOURCES\\r\\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\\r\\n"
    "\\r\\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\\r\\n"
    "#ifdef _WIN32\\r\\n"
    "LANGUAGE 9, 1\\r\\n"
    "#pragma code_page(1252)\\r\\n"
    "#endif\\r\\n"
    "#include \"res\\UC2Ani.rc2\" // non-Microsoft Visual C++ edited
resources\\r\\n"
    "#include \"afxres.rc\" // Standard components\\r\\n"
    "#endif\\0"
END

```

```
#endif    // APSTUDIO_INVOKED
```

```
#ifndef _MAC
```

```
////////////////////////////////////
//
// Version
//
```

```
VS_VERSION_INFO VERSIONINFO
```

```
FILEVERSION 2,1998,3,30
```

```
PRODUCTVERSION 2,0,0,0
```

```
FILEFLAGSMASK 0x3fL
```

```
#ifdef _DEBUG
```

```
FILEFLAGS 0x1L
```

```
#else
```

```
FILEFLAGS 0x0L
```

```
#endif
```

```
FILEOS 0x4L
```

```
FILETYPE 0x2L
```

```
FILESUBTYPE 0x0L
```

```
BEGIN
```

```
    BLOCK "StringFileInfo"
```

```
    BEGIN
```

```
        BLOCK "040904b0"
```

```
        BEGIN
```

```
            VALUE "Comments", "Special Thanks to mimi...\0"
```

```
            VALUE "CompanyName", "UNICHAT\0"
```

```
            VALUE "FileDescription", "UC2Ani DLL\0"
```

```
            VALUE "FileVersion", "2, 1998, 3, 30\0"
```

```
            VALUE "InternalName", "UC2Ani\0"
```

```
            VALUE "LegalCopyright", "Copyright (C) 1998 Programmed by
```

```
UNICHAT\0"
```

```
            VALUE "LegalTrademarks", "\0"
```

```
            VALUE "OriginalFilename", "UC2Ani.DLL\0"
```

```
            VALUE "PrivateBuild", "\0"
```

```
            VALUE "ProductName", "UC2Ani Dynamic Link Library\0"
```

```
            VALUE "ProductVersion", "2, 0, 0, 0\0"
```

```
            VALUE "SpecialBuild", "\0"
```

```
        END
```

```
    END
```

```
    BLOCK "VarFileInfo"
```

```
    BEGIN
```

```
        VALUE "Translation", 0x409, 1200
```

```
    END
```

```
END
```

```
#endif    // !_MAC
```

```
////////////////////////////////////
//
// Cursor
//
```

```
IDC_HARROW
```

```
CURSOR DISCARDABLE
```

```
"res\\harrow.cur"
```

```
#endif // English (U.S.) resources
////////////////////////////////////

#ifndef APSTUDIO_INVOKED
////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif
#include "res\UC2Ani.rc2" // non-Microsoft Visual C++ edited resources
#include "afxres.rc"      // Standard components
#endif
////////////////////////////////////
#endif // not APSTUDIO_INVOKED
```

```

//
//  CWave :
//
//  (C) Programmed by Kim
//
//  Information Technology Instituge
//  UNICHAT INC
//

#include "stdafx.h"
#include "wave.h"
#include "mem.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CWave

IMPLEMENT_SERIAL(CWave, COBJECT, 0 /* Schema number*/ )

// Create a simple waveform so there's something there.
CWave::CWave()
{
    m_pSamples = NULL;
    m_pOutDev = NULL;
    m_bBusy = FALSE;
    Create(16);
}

CWave::~CWave()
{
    if (m_bBusy)
    {
        Stop();
    }
    ASSERT(!m_bBusy);
    if (m_pSamples)
        FREE(m_pSamples);
}

////////////////////////////////////
// CWave serialization

void CWave::Serialize(CArchive& ar)
{
    ar.Flush();
    CFile* fp = ar.GetFile();

    if (ar.IsStoring())
    {
        ASSERT(0); // Save(fp);
    }
    else

```

```

    {
        Load(fp);
    }
}

////////////////////////////////////
// CWave notification functions
////////////////////////////////////

void CWave::OnWaveOutDone()
{
}

void CWave::OnWaveInData()
{
}

////////////////////////////////////
// CWave commands
////////////////////////////////////

BOOL CWave::Create(const int nsamples, const int samprate, const int
sampsizes)
{
    // validate the args
    if ((samprate != 11025) && (samprate != 22050) && (samprate != 44100))
    {
        TRACE("Invalid sample rate: %d", samprate);
        return FALSE;
    }
    if ((sampsizes != 8) && (sampsizes != 16))
    {
        TRACE("Invalid sample size: %d", sampsizes);
        return FALSE;
    }

    // Allocate memory for the samples.
    int iBytes = nsamples * sampsizes / 8;
    void* pSamples = ALLOC(iBytes);
    if (!pSamples)
    {
        TRACE("Out of memory for samples");
        return FALSE;
    }

    // Free existing memory and replace it.
    if (m_pSamples) FREE(m_pSamples);
    m_pSamples = pSamples;
    m_iSize = iBytes;

    // Fill out the format info.
    m_pcmfmt.wFormatTag          = WAVE_FORMAT_PCM;
    m_pcmfmt.nChannels           = 1; // We only do mono.
    m_pcmfmt.nSamplesPerSec      = samprate;
    m_pcmfmt.nAvgBytesPerSec     = samprate;
    m_pcmfmt.nBlockAlign         = sampsizes / 8; // Number of bytes
    m_pcmfmt.wBitsPerSample      = sampsizes;

    // Set the buffer to silence.

```

```

        for (int i=0; i < nsamples; i++)
        {
            SetSample(i, 0);
        }

        return TRUE;
    }

    BOOL CWave::Play(CWaveOutDevice* pWaveDevice)
    {
        if (pWaveDevice)
        {
            m_pOutDev = pWaveDevice;
            return pWaveDevice->Play(this);
        }
        else
        {
            m_pOutDev = &theDefaultWaveOutDevice;
            return theDefaultWaveOutDevice.Play(this);
        }
    }

    void CWave::Stop()
    {
        if (!m_bBusy)
            return;
        ASSERT(m_pOutDev);
        m_pOutDev->Stop();
    }

    BOOL CWave::Load(const char* pszFileName)
    {
        CString strFile;

        if ((pszFileName == NULL) || (strlen(pszFileName) == 0))
        {
            // Show an open file dialog to get the name.
            CFileDialog dlg(TRUE, // Open
                            NULL, // No default extension
                            NULL, // No initial file name
                            OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
                            "Wave files (*.WAV)|*.WAV|All files
(*.*)|*.*|");
            if (dlg.DoModal() == IDOK)
            {
                strFile = dlg.GetPathName();
            }
            else
            {
                return FALSE;
            }
        }
        else
        {
            // Copy the supplied file path.
            strFile = pszFileName;
        }
    }

```



```

// Try to open the file for read access.
CFile file;
if (!file.Open(strFile, CFile::modeRead | CFile::shareDenyWrite))
{
    AfxMessageBox("Failed to open file");
    return FALSE;
}

BOOL bResult = Load(&file);
file.Close();
if (!bResult) AfxMessageBox("Failed to load file");
return bResult;
}

BOOL CWave::Load(CFile *fp)
{
    return Load(fp->m_hFile);
}

BOOL CWave::Load(const UINT hFile)
{
    HMMIO hmmio;
    MMIOINFO info;
    ::ZeroMemory(&info, sizeof(info));
    info.adwInfo[0] = hFile;
    hmmio = mmioOpen(NULL, &info, MMIO_READ | MMIO_ALLOCBUF);
    if (!hmmio)
    {
        TRACE("mmioOpen failed");
        return FALSE;
    }
    BOOL bResult = Load(hmmio);
    mmioClose(hmmio, MMIO_FHOPEN);
    return bResult;
}

BOOL CWave::Load(const HMMIO hmmio)
{
    // Check whether it's a RIFF WAVE file.
    MMCKINFO ckFile;
    ckFile.fccType = mmioFOURCC('W', 'A', 'V', 'E');
    if (mmioDescend(hmmio, &ckFile, NULL, MMIO_FINDRIFF) != 0)
    {
        TRACE("Not a RIFF or WAVE file");
        return FALSE;
    }
    // Find the 'fmt ' chunk.
    MMCKINFO ckChunk;
    ckChunk.ckid = mmioFOURCC('f', 'm', 't', ' ');
    if (mmioDescend(hmmio, &ckChunk, &ckFile, MMIO_FINDCHUNK) != 0)
    {
        TRACE("No fmt chunk in file");
        return FALSE;
    }
    // Allocate some memory for the fmt chunk.
    int iSize = ckChunk.cksize;

```

```

WAVEFORMATEX* pfmt = (WAVEFORMATEX*)ALLOC(iSize);
ASSERT(pfmt);
// Read the fmt chunk.
if (mmioRead(hmmio, (char*)pfmt, iSize) != iSize)
{
    TRACE("Failed to read fmt chunk");
    FREE(pfmt);
    return FALSE;
}
// Check whether it's in PCM format.
if (pfmt->wFormatTag != WAVE_FORMAT_PCM)
{
    TRACE("Not a PCM file");
    FREE(pfmt);
    return FALSE;
}
// Get out of the fmt chunk.
mmioAscend(hmmio, &ckChunk, 0);
// Find the 'data' chunk.
ckChunk.ckid = mmioFOURCC('d','a','t','a');
if (mmioDescend(hmmio, &ckChunk, &ckFile, MMIO_FINDCHUNK) != 0)
{
    TRACE("No data chunk in file");
    FREE(pfmt);
    return FALSE;
}
// Allocate some memory for the data chunk.
iSize = ckChunk.cksize;
void* pdata = ALLOC(iSize);
if (!pdata)
{
    TRACE("No mem for data");
    FREE(pfmt);
    return FALSE;
}
// Read the data chunk.
if (mmioRead(hmmio, (char*)pdata, iSize) != iSize)
{
    TRACE("Failed to read data chunk");
    FREE(pfmt);
    FREE(pdata);
    return FALSE;
}
// Wrap the CWave object around what we have.
memcpy(&m_pcmfmt, pfmt, sizeof(m_pcmfmt));
// Replace the samples.
if (m_pSamples)
    FREE(m_pSamples);
m_pSamples = pdata;
m_iSize = iSize;

return TRUE;
}

BOOL CWave::LoadResource(const WORD wID)
{
    ASSERT(wID);

```

```

HINSTANCE hInst = AfxGetResourceHandle();
HRSRC hrsrc = ::FindResource(hInst, MAKEINTRESOURCE(WID), "WAVE");
if (!hrsrc)
{
    TRACE("WAVE resource not found");
    return FALSE;
}
HGLOBAL hg = ::LoadResource(hInst, hrsrc);
if (!hg)
{
    TRACE("Failed to load WAVE resource");
    return FALSE;
}
char* pRes = (char*)::LockResource(hg);
ASSERT(pRes);

// Mark the resource pages as read/write so the mmioOpen
// won't fail
int iSize = ::SizeofResource(hInst, hrsrc);
DWORD dwOldProt;
BOOL b = ::VirtualProtect(pRes, iSize, PAGE_READWRITE, &dwOldProt);
ASSERT(b);

// Open the memory block as an HMMIO object
HMMIO hmmio;
MMIOINFO info;
memset(&info, 0, sizeof(info));
info.fccIOProc = FOURCC_MEM;
info.pchBuffer = pRes;
info.cchBuffer = iSize;

hmmio = mmioOpen(NULL, &info, MMIO_READ);
if (!hmmio)
{
    TRACE("mmioOpen failed. Error %d\n", info.wErrorRet);
    return FALSE;
}
BOOL bResult = Load(hmmio);
mmioClose(hmmio, MMIO_FHOPEN);

// Note: not required to unlock or free the resource in Win32
return bResult;
}

// Get the number of samples.
int CWave::GetNumSamples() const
{
    ASSERT(m_pcmfmt.wBitsPerSample);
    return m_iSize * 8 / m_pcmfmt.wBitsPerSample;
}

// Get a sample value scaled as a 16 bit signed quantity.
int CWave::GetSample(int index) const
{
    if ((index < 0) || (index >= GetNumSamples()))
    {

```

```

        TRACE("Sample index out of range");
        return 0;
    }
    switch (m_pcmfmt.wBitsPerSample)
    {
    case 8:
    {
        BYTE *p = (BYTE *) m_pSamples;
        int i = p[index]; // 0 - 255;
        return (i - 128) * 256;
    }
    break;
    case 16:
    {
        ASSERT(sizeof(short int) == 2);
        short int* p = (short int *) m_pSamples;
        return p[index];
    }
    break;
    default:
        break;
    }
    ASSERT(1); // Invalid bits per sample
    return 0;
}

// Set a sample value from a 16 bit signed quantity.
void CWave::SetSample(const int index, const int iValue)
{
    if ((index < 0) || (index >= GetNumSamples()))
    {
        TRACE("Sample index out of range");
        return;
    }
    switch (m_pcmfmt.wBitsPerSample)
    {
    case 8:
    {
        BYTE* p = (BYTE*)m_pSamples;
        p[index] = iValue / 256 + 128;
    } break;

    case 16:
    {
        ASSERT(sizeof(short int) == 16);
        short int* p = (short int*)m_pSamples;
        p[index] = iValue;
    } break;

    default:
        ASSERT(1); // Invalid bits per sample
        break;
    }
}

```

```

//
//  CWave
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#ifndef __WAVE__
#define __WAVE__

#include "waveodev.h"

////////////////////////////////////
// CWave object

class AFX_EXT_CLASS CWave : public CObject
{
    DECLARE_SERIAL(CWave)
public:
    CWave();
    ~CWave();
    BOOL Create(const int nsamples, const int samprate=11025, const int
sampsiz=8);
    BOOL Play(CWaveOutDevice* pWaveOutDevice = NULL);
    void Stop();
    BOOL Load(const char* pszFileName=NULL);
    BOOL Load(CFile* fp);
    BOOL Load(const UINT hFile);
    BOOL Load(const HMMIO hmmio);
    BOOL LoadResource(const WORD wID);

// Attributes
public:
    WAVEFORMATEX* GetFormat() { return (WAVEFORMATEX*)&m_pcmfmt; }
    CWaveOutDevice* GetOutDevice() { return m_pOutDev; }
    int GetSize() const { return m_iSize; }
    int GetNumSamples() const;
    int GetSample(const int index) const;
    virtual void OnWaveOutDone();
    virtual void OnWaveInData();
    void SetSample(const int index, const int iValue);

// Implementation
public:
    void* GetSamples() { return m_pSamples; }
    BOOL IsBusy() const { return m_bBusy; }
    void SetBusy(const BOOL b) { m_bBusy = b; }

protected:
    virtual void Serialize(CArchive& ar); // Overridden for document
I/O
private:

```

UC2Ani\Wave.h

```
    WAVEFORMATEX    m_pcmfmt;    // PCM wave format header; PCMWAVEFORMAT-
->WAVEFORMATEX
    void*            m_pSamples; // Pointer to the samples
    int              m_iSize;     // Size in bytes
    CWaveOutDevice*  m_pOutDev;   // Output device
    BOOL             m_bBusy;     // Set to TRUE if playing or recording
};

#endif // __WAVE__
```

```

//
//  CWaveOutDevice :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "wave.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// global items
CWaveOutDevice theDefaultWaveOutDevice;

////////////////////////////////////
// CWaveOutDevice

CWaveOutDevice::CWaveOutDevice()
{
    m_hOutDev = NULL;
    m_iBlockCount = 0;
}

CWaveOutDevice::~CWaveOutDevice()
{
}

BEGIN_MESSAGE_MAP(CWaveOutDevice, CWnd)
    //{AFX_MSG_MAP(CWaveOutDevice)
    ON_MESSAGE(MM_WOM_DONE, OnWomDone)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CWaveOutDevice::Create()
{
    if (!CreateEx(0, AfxRegisterWndClass(0), "Wave Wnd",
        WS_POPUP, 0, 0, 0, 0, NULL, NULL))
    {
        TRACE("Failed to create wave notification window");
        return FALSE;
    }
    return TRUE;
}

#ifdef _DEBUG
#define MMERR(n) 0
#else
void MMERR(MMRESULT mmr)
{
    switch (mmr)
    {

```

```

    case WAVERR_BADFORMAT:
        TRACE("No wave device supports format");
        break;
    case MMSYSERR_NOMEM:
        TRACE("Out of memory");
        break;
    case MMSYSERR_ALLOCATED:
        TRACE("Resource is already allocated");
        break;
    case MMSYSERR_BADDEVICEID:
        TRACE("Bad device id");
        break;
    case MMSYSERR_INVALIDHANDLE:
        TRACE("Invalid device handle");
        break;
    case MMSYSERR_HANDLEBUSY:
        TRACE("Device in use by another thread");
        break;
    case WAVERR_UNPREPARED:
        TRACE("Header not prepared");
        break;
    default:
        TRACE("Unknown error");
        break;
}
#endif // _DEBUG

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CWaveOutDevice message handlers

LRESULT CWaveOutDevice::OnWomDone(WPARAM w, LPARAM l)
{
    ASSERT(l);
    WAVEHDR* phdr = (WAVEHDR*)l;
    CWave* pWave = (CWave*)(phdr->dwUser);
    ASSERT(pWave->IsKindOf(RUNTIME_CLASS(CWave)));
    CWaveOutDevice* pOutDev = pWave->GetOutDevice();
    ASSERT(pOutDev);
    pOutDev->WaveOutDone(pWave, phdr);
    return 0;
}

BOOL CWaveOutDevice::IsOpen()
{
    return m_hOutDev ? TRUE : FALSE;
}

// WAVEFORMAT superseded by WAVEFORMATEX in Visual C++ 4.0
BOOL CWaveOutDevice::Open(WAVEFORMATEX* pFormat)
{
    // Make sure we have a callback window.
    if (!m_hWnd)
    {
        Create(); // Create the window.
        ASSERT(m_hWnd);
    }
}

```



```

// See whether it's already open for this format.
if (IsOpen())
{
    // See whether it can handle this format.
    if (CanDoFormat(pFormat))
    {
        return TRUE;
    }
    else
    {
        TRACE("Open for different format");
        return FALSE;
    }
}

// See whether we can open for this format.
MMRESULT mmr = waveOutOpen(&m_hOutDev, WAVE_MAPPER, pFormat,
                           (DWORD)(GetSafeHwnd()), 0,
CALLBACK_WINDOW);
if (mmr)
{
    MMERR(mmr);
    return FALSE;
}

return TRUE;
}

BOOL CWaveOutDevice::CanDoFormat(WAVEFORMATEX* pFormat)
{
    if (!IsOpen())
    {
        TRACE("Not open");
        return FALSE;
    }
    HWAVEOUT hDev = NULL;
    MMRESULT mmr = waveOutOpen(&hDev, WAVE_MAPPER, pFormat, NULL, 0,
WAVE_FORMAT_QUERY);
    if (mmr)
    {
        MMERR(mmr);
        return FALSE;
    }
    return TRUE;
}

BOOL CWaveOutDevice::Close()
{
    if (m_hOutDev)
    {
        // Close the device.
        waveOutReset(m_hOutDev);
        MMRESULT mmr = waveOutClose(m_hOutDev);
        if (mmr)
            MMERR(mmr);
        m_hOutDev = NULL;
    }
}

```

```

    // Destroy the window.
    DestroyWindow();
    ASSERT(m_hWnd == NULL);
    return TRUE;
}

BOOL CWaveOutDevice::Play(CWave* pWave)
{
    if (!Open(pWave->GetFormat()))
        return FALSE;

    // Allocate a header.
    WAVEHDR* phdr = (WAVEHDR*)malloc(sizeof(WAVEHDR));
    ASSERT(phdr);
    // Fill out the wave header.
    ::ZeroMemory(phdr, sizeof(WAVEHDR));
    // #if 1 // BUGFIX for VC++ 2.0
    phdr->lpData = (BYTE*) pWave->GetSamples();
    // #else
    phdr->lpData = (char*) pWave->GetSamples();
    // #endif
    phdr->dwBufferLength = pWave->GetSize();
    phdr->dwUser = (DWORD) (void*) pWave; // So we can find the object

    // Prepare the header
    MMRESULT mmr = waveOutPrepareHeader(m_hOutDev, phdr, sizeof(WAVEHDR));
    if (mmr)
    {
        MMERR(mmr);
        return FALSE;
    }
    // Mark the wave as busy.
    pWave->SetBusy(TRUE);

    // Start it playing.
    mmr = waveOutWrite(m_hOutDev, phdr, sizeof(WAVEHDR));
    if (mmr)
    {
        MMERR(mmr);
        return FALSE;
    }

    // Add one to the block count.
    m_iBlockCount++;

    return TRUE;
}

void CWaveOutDevice::Stop()
{
    ASSERT(m_hOutDev);
    waveOutReset(m_hOutDev);
}

void CWaveOutDevice::WaveOutDone(CWave* pWave, WAVEHDR* pHdr)
{
    // Unprepare the header.

```

```
        MMRESULT mmr = waveOutUnprepareHeader(m_hOutDev, pHdr,
sizeof(WAVEHDR));
        if (mmr)
            MMERR(mmr);
        // Free the header.
        free(pHdr);

        // Decrement the block count.
        ASSERT(m_iBlockCount > 0);
        m_iBlockCount--;
        if (m_iBlockCount == 0) // Close the device.
            Close();

        // Notify the object that it is done.
        pWave->SetBusy(FALSE);
        pWave->OnWaveOutDone();
    }
```

```

//
//  CWaveOutDevice
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
#ifndef __WAVEODEV__
#define __WAVEODEV__

#include <mmsystem.h>

////////////////////////////////////
// CWaveOutDevice object

class CWave;

class AFX_EXT_CLASS CWaveOutDevice : public CWnd
{
// Attributes
public:
    BOOL  IsOpen();
    BOOL  CanDoFormat(WAVEFORMATEX* pFormat);

// Operations
public:
    CWaveOutDevice();
    BOOL  Open(WAVEFORMATEX* pFormat);
    BOOL  Close();
    BOOL  Play(CWave* pWave);
    void  Stop();
    void  WaveOutDone(CWave* pWave, WAVEHDR* pHdr);

// Implementation
public:
    virtual ~CWaveOutDevice();

private:
    BOOL  Create();

    HWAVEOUT  m_hOutDev;          // Output device handle
    int       m_iBlockCount;      // Number of blocks in the queue

    // Generated message map functions
protected:
    //{AFX_MSG(CWaveDevWnd)
    afx_msg LRESULT OnWomDone(WPARAM w, LPARAM l);
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// some global items
extern CWaveOutDevice theDefaultWaveOutDevice;

////////////////////////////////////

```

UC2Ani\WaveODev.h

#endif // __WAVEODEV__

11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

```

//
//  CDIB :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
#include "stdafx.h"
#include "DIB.h"
#include "DIBPal.h"

#include <lzexpand.h>

#ifdef _VICTOR
#include <vicdefs.h>    // link Vic32ms.lib
#endif

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
// #if !defined(KSM_REPORT)
// #define KSM_REPORT
// #endif
#endif

////////////////////////////////////
// CDIB

IMPLEMENT_SERIAL(CDIB, CObject, 0 /* Schema number */)

// Create a small DIB here so m_pBMI and m_pBits are always valid.
CDIB::CDIB()
{
    m_pBMI          = NULL;
    m_pBits          = NULL;
    m_bMyBits        = TRUE;
    m_bMapColorsDone = FALSE;
    m_strName.Empty();
    Create(16, 16);
}

CDIB::~CDIB()
{
    // Free the memory.
    if (m_pBMI)
        free(m_pBMI);
    if (m_bMyBits && m_pBits)
        free(m_pBits);
}

////////////////////////////////////
// CDIB serialization

// We don't support this yet.
void CDIB::Serialize(CArchive& ar)
{

```

```

        ar.Flush();
        CFile* fp = ar.GetFile();

        if (ar.IsStoring())
        {
            Save(fp);
        }
        else
        {
            Load(fp);
        }
    }

    //////////////////////////////////////
    // Private functions

static BOOL IsWinDIB(const BITMAPINFOHEADER *pBIH)
{
    ASSERT(pBIH);
    if (((BITMAPCOREHEADER*)pBIH)->bcSize == sizeof(BITMAPCOREHEADER))
        return FALSE;
    return TRUE;
}

static int NumDIBColorEntries(BITMAPINFO* pBmpInfo)
{
    BITMAPINFOHEADER* pBIH;
    BITMAPCOREHEADER* pBCH;
    int iColors, iBitCount;

    ASSERT(pBmpInfo);

    pBIH = &(pBmpInfo->bmiHeader);
    pBCH = (BITMAPCOREHEADER*)pBIH;

    // Start off by assuming the color table size from the bit-per-pixel
field.
    iBitCount = IsWinDIB(pBIH) ? pBIH->biBitCount : pBCH->bcBitCount;

    switch (iBitCount)
    {
        case 1:          iColors = 2;          break;
        case 4:          iColors = 16;         break;
        case 8:          iColors = 256;        break;
        default:         iColors = 0;          break;
    }

    // If this is a Windows DIB, then the color table length is determined
    // by the biClrUsed field if the value in the field is nonzero.
    if (IsWinDIB(pBIH) && (pBIH->biClrUsed != 0))
        iColors = pBIH->biClrUsed;

    // BUGFIX 18 Oct 94 Nigelt
    // Make sure the value is reasonable since some products
    // will write out more than 256 colors for an 8 bpp DIB!!!
    int iMax = 0;
    switch (iBitCount)

```

```

    {
    case 1:          iMax = 2;   break;
    case 4:          iMax = 16;  break;
    case 8:          iMax = 256; break;
    default:         iMax = 0;   break;
    }
    if (iMax)
    {
        if (iColors > iMax)
        {
            TRACE("Invalid color count\n");
            iColors = iMax;
        }
    }
    return iColors;
}

/////////////////////////////////////////////////////////////////
// CDIB commands
#ifdef _DEBUG

#endif

// Create a new empty 8bpp DIB with a 256 entry color table.
BOOL CDIB::Create(const iWidth, const iHeight, const char* pszPalFileName)
{
    // Delete any existing stuff.
    if (m_pBMI)
        free(m_pBMI);
    if (m_bMyBits && m_pBits)
        free(m_pBits);

    // Allocate memory for the header.
    m_pBMI = (BITMAPINFO*)malloc(sizeof(BITMAPINFOHEADER) + 256 *
sizeof(RGBQUAD));
    if (!m_pBMI)
    {
        TRACE("Out of memory for DIB header\n");
        return FALSE;
    }

    // Allocate memory for the bits (DWORD aligned).
    int iBitsSize = ((iWidth + 3) & ~3) * iHeight;
    m_pBits = (BYTE*)malloc(iBitsSize);
    if (!m_pBits)
    {
        TRACE("Out of memory for DIB bits\n");
        free(m_pBMI);
        m_pBMI = NULL;
        return FALSE;
    }
    m_bMyBits = TRUE;

    // Fill in the header info.
    BITMAPINFOHEADER* pBI = (BITMAPINFOHEADER*)m_pBMI;
    pBI->biSize      = sizeof(BITMAPINFOHEADER);
    pBI->biWidth      = iWidth;
    pBI->biHeight     = iHeight;

```



```

    pBI->biPlanes      = 1;
    pBI->biBitCount     = 8;
    pBI->biCompression  = BI_RGB;
    pBI->biSizeImage    = 0;
    pBI->biXPelsPerMeter = 0;
    pBI->biYPelsPerMeter = 0;
    pBI->biClrUsed      = 0;
    pBI->biClrImportant = 0;

    // Create an arbitrary color table (gray scale).
    if (pszPalFileName)
    {
        LoadPalette(pszPalFileName);
    }
    else
    {
        RGBQUAD* prgb = GetClrTabAddress();
        for (int i=0; i < 256; i++)
        {
            prgb->rgbBlue = prgb->rgbGreen = prgb->rgbRed = (BYTE)i;
            prgb->rgbReserved = 0;
            prgb++;
        }
    }
    // Set all the bits to a known state (black).
    ::ZeroMemory(m_pBits, iBitsSize);

    return TRUE;
}

int CDIB::GetBitsSize() const
{
    if (!m_pBMP)
        return 0;
    BITMAPINFOHEADER* pBI = (BITMAPINFOHEADER*)m_pBMP;
    return (((pBI->biWidth + 3) & ~3)*pBI->biHeight);
}

void CDIB::ClearImage()
{
    if (m_pBits)
    {
        ::ZeroMemory(m_pBits, GetBitsSize());
    }
}

void CDIB::ClearRect(CRect& rcClear)
{
    if (!m_pBits)
        return;
    int w = rcClear.Width();
    int h = rcClear.Height();
    // Test for silly cases.
    if (w == 0 || h == 0)
        return;
    int xd = rcClear.left;
    int yd = rcClear.top;

```

```

BYTE* pDest = (BYTE*)GetPixelAddress(xd, yd + h - 1);
ASSERT(pDest);

// Get the scan line widths of each DIB.
int iDInc = StorageWidth(); // Source increment value
while (h--) // Fill the lines
{
    ::ZeroMemory(pDest, w);
    pDest += iDInc;
}

// Create a CDIB structure from existing header and bits.
// The DIB won't delete the bits and makes a copy of the header.
BOOL CDIB::Create(BITMAPINFO* pBMI, BYTE* pBits)
{
    ASSERT(pBMI);
    ASSERT(pBits);
    if (m_pBMI)
        free(m_pBMI);
    m_pBMI = (BITMAPINFO*)malloc(sizeof(BITMAPINFOHEADER) +
256*sizeof(RGBQUAD));
    ASSERT(m_pBMI);
    // Note: This will probably fail for < 256 color headers.
    ::CopyMemory(m_pBMI, pBMI, sizeof(BITMAPINFOHEADER) +
NumDIBColorEntries(pBMI)*sizeof(RGBQUAD));

    if (m_bMyBits && m_pBits)
        free(m_pBits);
    m_pBits = pBits;
    m_bMyBits = FALSE; // We can't delete the bits.
    return TRUE;
}

// Load a palette from the file and replace existing one
BOOL CDIB::LoadPalette(const char* pszPalFileName)
{
    CDIBPal pal;
    PALETTEENTRY PE[256];
    BOOL bRes = pal.Load(pszPalFileName, PE);
    SetPaletteEntries(0, 256, PE);
    return bRes;
}

// Load a DIB from an open file.
// Soomin Kim made the following BMP variations
// BMZ: LZ compressed BMP file
// BM: Excluding palette section from BMP
// BMC: Compressed BM file
BOOL CDIB::Load(CFile* const fp, const char* pszPalFileName)
{
    BOOL bIsPM = FALSE;
    BOOL bIsLZ = FALSE;
    BOOL bIsBMC = FALSE;
    BITMAPINFO* pBmpInfo = NULL;
    BYTE* pBits = NULL;
    UINT hLZFile;

```

```

// Get the current file position.
DWORD dwFileStart = fp->GetPosition();

// Read the file header to get the file size and to find where the bits
start in the file.
BITMAPFILEHEADER BmpFileHdr;
int iBytes = fp->Read(&BmpFileHdr, sizeof(BmpFileHdr));
if (iBytes != sizeof(BmpFileHdr))
{
    TRACE("Failed to read file header\n");
    goto $abort;
}

if (BmpFileHdr.bfType == 0x5a53)    // 'SZ'
{
    bIsLZ = TRUE;
    hLZFile = ::LZInit(fp->m_hFile);
    // long lFileSize = ::LZSeek(hFile, 0L, 2); // points to the end of
file
    // TRACE("Compressed BMP: %ld bytes.\n", lFileSize);
    ::LZSeek(hLZFile, dwFileStart, 0); // seek to beginning of file
    ::LZRead(hLZFile, (char*)&BmpFileHdr, sizeof(BmpFileHdr));
}

if (BmpFileHdr.bfType == 0x4342)    // 'BC'
{
    bIsBMC = TRUE;
}
else if (BmpFileHdr.bfType != 0x4d42)    // 'BM'
{
    TRACE("Not a bitmap file\n");
    goto $abort;
}

// Make a wild guess that the file is in Windows DIB format and read
the BITMAPINFOHEADER.
// If the file turns out to be a PM DIB file we'll convert it later.
BITMAPINFOHEADER BmpInfoHdr;
if (bIsLZ)
    iBytes = ::LZRead(hLZFile, (char*)&BmpInfoHdr,
sizeof(BmpInfoHdr));
else
    iBytes = fp->Read(&BmpInfoHdr, sizeof(BmpInfoHdr));
if (iBytes != sizeof(BmpInfoHdr))
{
    TRACE("Failed to read BITMAPINFOHEADER\n");
    goto $abort;
}

// Check that we got a real Windows DIB file.
if (BmpInfoHdr.biSize != sizeof(BITMAPINFOHEADER))
{
    if (BmpInfoHdr.biSize != sizeof(BITMAPCOREHEADER))
    {
        TRACE(" File is not Windows or PM DIB format\n");
        goto $abort;
    }
}

```

```

    }

    // Set a flag to convert PM file to Win format later.
    bIsPM = TRUE;

    // Back up the file pointer and read the BITMAPCOREHEADER
    // and create the BITMAPINFOHEADER from it.
    if (bIsLZ)
        ::LZSeek(hLZFile, dwFileStart + sizeof(BITMAPFILEHEADER),
0);
    else
        fp->Seek(dwFileStart + sizeof(BITMAPFILEHEADER),
CFile::begin);
    BITMAPCOREHEADER BmpCoreHdr;
    if (bIsLZ)
        iBytes = ::LZRead(hLZFile, (char*)&BmpCoreHdr,
sizeof(BmpCoreHdr));
    else
        iBytes = fp->Read(&BmpCoreHdr, sizeof(BmpCoreHdr));
    if (iBytes != sizeof(BmpCoreHdr))
    {
        TRACE("Failed to read BITMAPCOREHEADER\n");
        goto $abort;
    }

    BmpInfoHdr.biSize           = sizeof(BITMAPINFOHEADER);
    BmpInfoHdr.biWidth          = (int)BmpCoreHdr.bcWidth;
    BmpInfoHdr.biHeight         = (int)BmpCoreHdr.bcHeight;
    BmpInfoHdr.biPlanes         = BmpCoreHdr.bcPlanes;
    BmpInfoHdr.biBitCount       = BmpCoreHdr.bcBitCount;
    BmpInfoHdr.biCompression    = BI_RGB;
    BmpInfoHdr.biSizeImage      = 0;
    BmpInfoHdr.biXPelsPerMeter   = 0;
    BmpInfoHdr.biYPelsPerMeter  = 0;
    BmpInfoHdr.biClrUsed        = 0;
    BmpInfoHdr.biClrImportant   = 0;
}

// Work out how much memory we need for the BITMAPINFO structure, color
table
// and then for the bits. Allocate the memory blocks.
// Copy the BmpInfoHdr we have so far, and then read in the color table
from the file.
int iColors;
int iColorTableSize;
iColors =
NumDIBColorEntries((LPBITMAPINFO)&BmpInfoHdr);
iColorTableSize = iColors * sizeof(RGBQUAD);
// Always allocate enough room for 256 entries.
int iBISize;
int iBitsSize;
iBISize = sizeof(BITMAPINFOHEADER) + 256 * sizeof(RGBQUAD);
iBitsSize = BmpFileHdr.bfSize - BmpFileHdr.bfOffBits;

TRACE("CDIB::Load(\"%s\") size=%ld\n", m_strName, BmpFileHdr.bfSize);

// Allocate the memory for the header.

```

```

// =====
pBmpInfo = (LPBITMAPINFO)malloc(iBISize);
if (!pBmpInfo)
{
    TRACE("Out of memory for DIB header\n");
    goto $abort;
}

// Copy the header we already have.
::CopyMemory(pBmpInfo, &BmpInfoHdr, sizeof(BITMAPINFOHEADER));

// Now read the color table from the file.
if (bIsBMC)
{
    ::ZeroMemory(((LPBYTE)pBmpInfo) + sizeof(BITMAPINFOHEADER),
iColorTableSize);
}
else
{
    if (!bIsPM)
    {
        // Read the color table from the file.
        if (bIsLZ)
            iBytes = ::LZRead(hLZFile, (char*)((LPBYTE)pBmpInfo)
+ sizeof(BITMAPINFOHEADER)), iColorTableSize);
        else
            iBytes = fp->Read(((LPBYTE)pBmpInfo) +
sizeof(BITMAPINFOHEADER), iColorTableSize);
        if (iBytes != iColorTableSize)
        {
            TRACE("Failed to read color table\n");
            goto $abort;
        }
    }
    else
    {
        // Read each PM color table entry in turn and convert it to
Win DIB format as we go.
        LPRGBQUAD lpRGB = (LPRGBQUAD)((LPBYTE)pBmpInfo +
sizeof(BITMAPINFOHEADER));
        RGBTRIPLE rgbt;
        for (int i=0; i < iColors; i++)
        {
            if (bIsLZ)
                iBytes = ::LZRead(hLZFile, (char*)&rgbt,
sizeof(RGBTRIPLE));
            else
                iBytes = fp->Read(&rgbt, sizeof(RGBTRIPLE));
            if (iBytes != sizeof(RGBTRIPLE))
            {
                TRACE("Failed to read RGBTRIPLE\n");
                goto $abort;
            }
            lpRGB->rgbBlue    = rgbt.rgbtBlue;
            lpRGB->rgbGreen   = rgbt.rgbtGreen;
            lpRGB->rgbRed     = rgbt.rgbtRed;
            lpRGB->rgbReserved = 0;
        }
    }
}

```

```

        lpRGB++;
    }
}

// Allocate the memory for the bits and read the bits from the file.
// =====
pBits = (BYTE*)malloc(iBitsSize);
if (!pBits)
{
    TRACE("Out of memory for DIB bits\n");
    goto $abort;
}

// Seek to the bits in the file.
if (bIsLZ)
    ::LZSeek(hLZFile, dwFileStart + BmpFileHdr.bfOffBits, 0);
else
    fp->Seek(dwFileStart + BmpFileHdr.bfOffBits, CFile::begin);

// Read the bits.
if (bIsLZ)
    iBytes = ::LZRead(hLZFile, (char*)pBits, iBitsSize);
else
    iBytes = fp->Read(pBits, iBitsSize);
if (iBytes != iBitsSize)
{
    TRACE("Failed to read bits\n");
    goto $abort;
}

// Everything went OK.
if (bIsLZ)
    ::LZClose(hLZFile);
if (m_pBMI)
    free(m_pBMI);
m_pBMI = pBmpInfo;
if (m_bMyBits && m_pBits)
    free(m_pBits);
m_pBits = pBits;
m_bMyBits = TRUE;

if (pszPalFileName) // Overload the palette entries with that of
the specified file
{
    LoadPalette(pszPalFileName);
}
return TRUE;

$abort: // Something went wrong.
if (bIsLZ)
    ::LZClose(hLZFile);
if (pBmpInfo)
    free(pBmpInfo);
if (pBits)
    free(pBits);
return FALSE;
}

// Load a DIB from a disk file.

```

```

// If no file name is given, show an Open File dialog to get one.
BOOL CDIB::Load(const char* pszFileName, const char* pszPalFileName)
{
    if ((pszFileName == NULL) || (strlen(pszFileName) == 0))
    {
        // Show an Open File dialog to get the name.
        CFileDialog dlg(TRUE, // Open
            NULL, // No default extension
            NULL, // No initial file name
            OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,

#ifdef _VICTOR
            "Bitmap
files|.DIB|.BMP|.BM|.gif|.jpg|.tif|All files (*.*)|*.*|");
#else
            "Bitmap
files|.DIB|.BM*|.DIB|.BM*|All files (*.*)|*.*|");
#endif // _VICTOR
        if (dlg.DoModal() == IDOK)
            m_strName = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        m_strName = pszFileName;
    }

    BOOL bResult;
#ifdef _VICTOR
    int nFT = GetBitmapFileType(m_strName);
    if ((nFT == FILE_BMP) || (nFT == FILE_BM) || (nFT == FILE_BMZ) || (nFT
== FILE_BMC))
    {
#endif // _VICTOR
        // Try to open the file for read access.
        CFile file;
        if (!file.Open(m_strName, CFile::modeRead |
CFile::shareDenyWrite))
        {
            TRACE1("File Open Failure: %s\n", m_strName);
            return FALSE;
        }
        bResult = Load(&file, pszPalFileName);
        file.Close();
#ifdef _VICTOR
    }
    else
    {
        imgdes orgImg; // original image
        imgdes cnvImg; // converted image (24bit color -> 8bit color)
        switch (nFT)
        {
        case FILE_GIF:
            LoadGIF(m_strName, &orgImg);
            break;
        case FILE_TIF:

```

```

        LoadTIF(m_strName, &orgImg);
        break;
    case FILE_JPG:
        LoadJPG(m_strName, &orgImg);
        break;
    }
    if (orgImg.bmh->biBitCount != 8)
    {
        if (CreateImage8(&orgImg, &cnvImg) != NO_ERROR)
        {
            AfxMessageBox("Image color dithering failure.");
            return FALSE;
        }
        freeimage(&orgImg);
        bResult = LoadFromImage((LPVOID)cnvImg.bmh);
        freeimage(&cnvImg);
    }
    else
    {
        bResult = LoadFromImage((LPVOID)orgImg.bmh);
        freeimage(&orgImg);
    }
}
#endif // _VICTOR

    return bResult;
}

BOOL CDIB::LoadSource(const char* pszSrcFileName,
                    const char* pszFileName,
                    const char* pszPalFileName)
{
    // Open pszSrcFileName
    //
    /*
        CFile file;
        if (!file.Open(m_strName, CFile::modeRead |
CFile::shareDenyWrite))
        {
            TRACE1("File Open Failure: %s\n", m_strName);
            return FALSE;
        }
        ::LZSeek(hLZFile, dwFileStart + BmpFileHdr.bfOffBits, 0);
        bResult = Load(&file, pszPalFileName);
    */
    return TRUE;    // Load DIB from disk file
}

// Load a DIB from a resource id.
BOOL CDIB::Load(const WORD wResid)
{
    ASSERT(wResid);
    HINSTANCE hInst = AfxGetResourceHandle();
    HRSRC hrsrc = ::FindResource(hInst, MAKEINTRESOURCE(wResid), "DIB");
    if (!hrsrc)
    {
        TRACE1("DIB resource not found(HRSRC:%x)", hrsrc);
    }
}

```



```

        return FALSE;
    }
    HGLOBAL hg = ::LoadResource(hInst, hrsrc);
    if (!hg)
    {
        TRACE1("Failed to load DIB resource(HGLOBAL:%x)", hg);
        return FALSE;
    }

    m_strName.Format("RES:%d", wResid);

    BYTE* pRes = (BYTE*)::LockResource(hg);
    ASSERT(pRes);
    int iSize = ::SizeofResource(hInst, hrsrc);

    // Mark the resource pages as read/write so the mmioOpen won't fail
    DWORD dwOldProt;
    BOOL b = ::VirtualProtect(pRes, iSize, PAGE_READWRITE, &dwOldProt);
    ASSERT(b);

    // Now create the CDIB object. We will create a new header from the
data    // in the resource image and copy the bits from the resource to a new
block    // of memory.
    // We can't use the resource image as-is because we might want to map
the DIB    // the DIB colors
    // and the resource memory is write protected in Win32.
    BITMAPFILEHEADER* pFileHdr = (BITMAPFILEHEADER*)pRes;
    ASSERT(pFileHdr->bfType == 0x4D42); // BM file
    BITMAPINFOHEADER* pInfoHdr = (BITMAPINFOHEADER*)(pRes +
sizeof(BITMAPFILEHEADER));
    ASSERT(pInfoHdr->biSize == sizeof(BITMAPINFOHEADER)); // must be a Win
DIB
    BYTE* pBits = pRes + pFileHdr->bfOffBits;
    BOOL bResult = Create((BITMAPINFO*)pInfoHdr, pBits);
    return bResult;
    // Note: not required to unlock or free the resource in Win32
}

// Draw the DIB to a given DC.
void CDIB::Draw(CDC* pDC, const x, const y)
{
    ::StretchDIBits(pDC->GetSafeHdc(),
        x, y, DibWidth(), DibHeight(), // Destination x, y,
width, height    0, 0, DibWidth(), DibHeight(), // Source x, y, width,
height          GetBitsAddress(), // Pointer to bits
                GetBitmapInfoAddress(), // BITMAPINFO
                DIB_RGB_COLORS, // Options
                SRCCOPY); // Raster
operation code (ROP)
}

// Draw a portion of source DIB to the destination
void CDIB::Draw(CDC* pDC, const xd, const yd, const w, int h, const xs, const
ys)

```

```

{
    ::StretchDIBits(pDC->GetSafeHdc(),
                    xd, yd, w, h,
                    // Destination x,
y, width, height
                    xs, DibHeight()-(ys+h), w, h, // Source x, y, width, height
                    GetBitsAddress(),
                    // Pointer to bits
                    GetBitmapInfoAddress(),
                    // BITMAPINFO
                    DIB_RGB_COLORS,
                    // Options
                    SRCCOPY);
                    // Raster
operation code (ROP)
}

// Tile
void CDIB::Tile(CDC* pDC, const x0, const y0, CRect& rcClient)
{
    for (int y=y0; y <= rcClient.bottom; y += DibHeight())
        for (int x=x0; x <= rcClient.right; x += DibWidth())
            Draw(pDC, x, y);
}

// Get the number of color table entries.
int CDIB::GetNumClrEntries() const
{
    return NumDIBColorEntries(m_pBMI);
}

// map the colors in this DIB to the identity palette specified
// NOTE: This assumes all CDIB objects have 256 color table entries.
BOOL CDIB::MapColorsToPalette(const CPalette* pPal)
{
    if (m_bMapColorsDone)
        return TRUE;
    if (!pPal)
    {
        TRACE("No palette to map to\n");
        return FALSE;
    }
    ASSERT(m_pBMI);
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    ASSERT(m_pBits);
    LPRGBQUAD pctThis = GetClrTabAddress(); // We can redirect the
palette address
// with
SetClrTabAddress(...) function.
    ASSERT(pctThis);
    // Build an index translation table to map this DIBs colors to those of
the reference DIB.
    BYTE imap[256];
#ifdef _DEBUG
    int iChanged = 0; // For debugging only
#endif
    for (int i=0; i < 256; i++)
    {
        imap[i] = (BYTE)pPal->GetNearestPaletteIndex(
            RGB(pctThis->rgbRed,
                pctThis->rgbGreen,
                pctThis->rgbBlue));
    }
}

```

```

        pctThis++;
#ifdef _DEBUG
        if (imap[i] != i)
            iChanged++; // For debugging
    }
    TRACE("CDIB::MapColorsToPalette changed %d colors.\n", iChanged);
#else
    }
#endif
// Now map the DIB bits.
BYTE* pBits = (BYTE*)GetBitsAddress();
int iSize = StorageWidth() * DibHeight();
while (iSize--)
{
    *pBits = imap[*pBits];
    pBits++;
}
// Now reset the DIB color table so that its RGB values match those in
the palette.
PALETTEENTRY pe[256];
pPal->GetPaletteEntries(0, 256, pe);
pctThis = GetClrTabAddress();
for (i=0; i < 256; i++)
{
    pctThis->rgbRed      = pe[i].peRed;
    pctThis->rgbGreen    = pe[i].peGreen;
    pctThis->rgbBlue     = pe[i].peBlue;
    pctThis++;
}
// Now say all the colors are in use
m_pBMI->bmiHeader.biClrUsed = 256;
m_bMapColorsDone = TRUE;
return TRUE;
}

// Get a pointer to a pixel.
// NOTE: DIB scan lines are DWORD aligned.
// The scan line storage width may be wider than the scan line image width
// so calc the storage width by rounding the image width to the next highest
DWORD value.
void* CDIB::GetPixelAddress(const x, const y) const
{
    // Note: This version deals only with 8 bpp DIBs.
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    // Make sure it's in range and if it isn't return zero.
    if ((x >= DibWidth()) || (y >= DibHeight()))
    {
        TRACE("Attempt to get out of range pixel address\n");
        return NULL;
    }

    // Calculate the scan line storage width.
    int iWidth = StorageWidth();
    return m_pBits + (DibHeight()-y-1) * iWidth + x;
}

// Get the bounding rectangle.

```

```

void CDIB::GetRect(CRect& rect) const
{
    rect.left = rect.top = 0;
    rect.right = DibWidth();
    rect.bottom = DibHeight();
}

// Copy a rectangle of the DIB to another DIB.
// Note: We only support 8bpp DIBs here.
void CDIB::CopyBits(CDIB* pdibDest, const xd, const yd,
                    const w, int h, const xs, const ys, const COLORREF
clrTrans,
                    const WORD wImOp)
{
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    ASSERT(pdibDest);
    WORD opacity = wImOp & OPACITY_MASK;
    // Test for silly cases.
    if (w == 0 || h == 0 || opacity == OPACITY_0)
        return;

    BOOL bMirror = wImOp & IMAGE_FLIP;
    BOOL bVT = wImOp & IMAGE_VERTICAL; // Vertical Transformation

    // Get pointers to the start points in the source and destination DIBs.
    // Note that the start points will be the bottom-left corner of the
DIBs
    // because the scan lines are reversed in memory.
    BYTE* pSrc = (BYTE*)GetPixelAddress(xs, ys+h-1);
    ASSERT(pSrc);
    BYTE* pDest = (BYTE*)pdibDest->GetPixelAddress(xd, (bVT) ? yd : (yd+h-
1));
    ASSERT(pDest);

    // Get the scan line widths of each DIB.
    int iScanS = StorageWidth();
    int iScanD = pdibDest->StorageWidth();
    int iSInc = (bMirror) ? iScanS : (iScanS - w); // Source increment
value
    int iDInc = (bVT) ? (-iScanD - w) : (iScanD - w); // Destination
increment value
    int iCount;
    BYTE pixel;

    if (clrTrans == 0xFFFFFFFF) // && opacity == OPACITY_100)
    {
        if (bMirror)
        {
            while (h--) // Copy the lines
            {
                iCount = w; // Number of pixels to scan.
                pSrc += w;
                while (iCount--)
                {
                    pixel = *(--pSrc);
                    *pDest++ = pixel;
                }
            }
        }
    }
}

```

```

        pSrc += iSInc;
        pDest += iDInc;
    }
}
else // !Mirror
{
    while (h--) // Copy the lines
    {
        ::CopyMemory(pDest, pSrc, w);
        pSrc += iScanS;
        if (bVT)
            pDest -= iScanD;
        else
            pDest += iScanD;
    }
}
}
else // USE_COLORKEY
{
    // Copy lines with transparency.
    // Note: We accept only a PALETTEINDEX description for the color
definition.
    ASSERT((clrTrans & 0xFF000000) == 0x01000000);
    BYTE bTransClr = LOBYTE(LOWORD(clrTrans));

    if (opacity == OPACITY_100)
    {
        if (bMirror)
        {
            while (h--)
            {
                iCount = w;    // Number of pixels to scan.
                pSrc += w;
                while (iCount--)
                {
                    pixel = *(--pSrc);          // *pSrc--
was THE BUG!!! 22:30 Oct 12 Sat '96 at ImKwang
transparent.
                    // Copy pixel only if it isn't

                    if (pixel != bTransClr) *pDest++ = pixel;
                    else
                        pDest++;
                }
                // Move on to the next line.
                pSrc += iSInc;
                pDest += iDInc;
            }
        }
        else // !bMirror
        {
            while (h--)
            {
                iCount = w;    // Number of pixels to scan.
                while (iCount--)
                {
                    pixel = *pSrc++;
transparent.
                    // Copy pixel only if it isn't

```

```

        if (pixel != bTransClr) *pDest++ = pixel;
        else                    pDest++;
    }
    // Move on to the next line.
    pSrc += iSInc;
    pDest += iDInc;
}
}
}
else if (opacity == OPACITY_75)
{
    int r;        // zero or one
    int d=4;      // 3/4

    if (bMirror)
    {
        while (h--)
        {
            iCount = w;    // Number of pixels to scan.
            r = h % d;      // 0,1,2,3,0,... =>

            if (r==1)
                r = 2;
            else if (r==2)
                r = 1;
            pSrc += w;
            while (iCount--)
            {
                pixel = *--pSrc;
                // Copy pixel only if it isn't
                // transparent.
                if ((iCount % d != r) && (pixel !=
                bTransClr))
                {
                    *pDest++ = pixel;
                }
                else
                    pDest++;
            }
            // Move on to the next line.
            pSrc += iSInc;
            pDest += iDInc;
        }
    }
    else // !bMirror
    {
        while (h--)
        {
            iCount = w;    // Number of pixels to scan.
            r = h % d;
            if (r==1)
                r = 2;
            else if (r==2)
                r = 1;
            while (iCount--)
            {
                pixel = *pSrc++;
                // Copy pixel only if it isn't
                // transparent.
            }
        }
    }
}

```

```

        if ((iCount % d != r) && (pixel !=
bTransClr))
            *pDest++ = pixel;
        else
            pDest++;
    }
    // Move on to the next line.
    pSrc += iSInc;
    pDest += iDInc;
}
}
else // !OPACITY_100, OPACITY_75
{
    int r;
    int d;

    switch (opacity)
    {
        case OPACITY_12: d = 4; break; // 1/64 paint
        case OPACITY_25: // 1/4
        case OPACITY_50: d = 2; break; // 1/2
        default: d = 2; break;
    }

    if (bMirror)
    {
        while (h--)
        {
            iCount = w; // Number of pixels to scan.
            r = h % d; // Since adjacent lines should be
different
            pSrc += w;
            while (iCount--)
            {
                if ((opacity != OPACITY_50) && (r != 0))
                {
                    pSrc -= (iCount+1);
                    pDest += (iCount+1);
                    break; // skip this scan
line
                }
                pixel = *--pSrc;
                // Copy pixel only if it isn't
transparent.
                if ((iCount % d == r) && (pixel !=
bTransClr))
                    *pDest++ = pixel;
                else
                    pDest++;
            }
            // Move on to the next line.
            pSrc += iSInc;
            pDest += iDInc;
        }
    }
    else // !Mirror

```

```

    {
        while (h--)
        {
            iCount = w;    // Number of pixels to scan.
            r = h % d;
            while (iCount--)
            {
                if ((opacity != OPACITY_50) && (r != 0))
                {
                    pSrc += (iCount+1);    // since 1
                    pDest += (iCount+1);
                    break;    // skip this scan
                }
                pixel = *pSrc++;
                // Copy pixel only if it isn't
                if ((iCount % d == r) && (pixel !=
                    *pDest++ = pixel;
                else
                    pDest++;
            }
            // Move on to the next line.
            pSrc += iSInc;
            pDest += iDInc;
        }
    }
}

#ifdef KSM_REPORT
    static int D_CopyBitsCount = 0;
    TRACE3("CDIB::CopyBits(%d,%d)=%d\n", w, h, ++D_CopyBitsCount);
#endif
}

// Save a DIB to a disk file.
// This is somewhat simplistic because we only deal with 256 color DIBs
// and we always write a 256 color table.
BOOL CDIB::Save(CFile* const fp, const BOOL bPalette)
{
    BITMAPFILEHEADER bfh;

    // Construct the file header.
    bfh.bfType = (bPalette) ? 0x4D42 : 0x4342; // 'BM' or 'BC'
    bfh.bfSize =
        sizeof(BITMAPFILEHEADER) +
        sizeof(BITMAPINFOHEADER) +
        StorageWidth() * DibHeight();
    if (bPalette)
        bfh.bfSize += 256 * sizeof(RGBQUAD);
    bfh.bfReserved1 = 0;
    bfh.bfReserved2 = 0;
    bfh.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
    if (bPalette)
        bfh.bfOffBits += 256 * sizeof(RGBQUAD);
}

```



```

// Write the file header.
int iSize = sizeof(bfh);
TRY
{
    fp->Write(&bfh, iSize);
}
CATCH(CFileException, e)
{
    TRACE("Failed to write file header");
    return FALSE;
} END_CATCH

// Write the BITMAPINFO structure.
// Note: we assume that there are always 256 colors in the color table.
ASSERT(m_pBMI);
iSize = sizeof(BITMAPINFOHEADER);
if (bPalette)
    iSize += 256 * sizeof(RGBQUAD);
TRY
{
    fp->Write(m_pBMI, iSize);
}
CATCH(CFileException, e)
{
    TRACE("Failed to write BITMAPINFO");
    return FALSE;
} END_CATCH

// Write the bits.
iSize = StorageWidth() * DibHeight();
TRY
{
    fp->Write(m_pBits, iSize);
}
CATCH(CFileException, e)
{
    TRACE("Failed to write bits");
    return FALSE;
} END_CATCH

return TRUE;
}

// Save a DIB to a disk file. If no file name is given, show a File Save
// dialog to get one.
BOOL CDIB::Save(const char* pszFileName)
{
    CString strFile;

    if ((pszFileName == NULL) || (strlen(pszFileName) == 0))
    {
        // Show a File Save dialog to get the name.
        CFileDialog dlg(FALSE, // Save
                        NULL, // No default extension
                        m_strName, // No initial file name
                        OFN_OVERWRITEPROMPT | OFN_HIDEREADONLY,

```

```

                                "Image files
(*.DIB;*.BM*)|*.DIB;*.BM*|All files (*.*)|*.*||");
    if (dlg.DoModal() == IDOK)
    {
        strFile = dlg.GetPathName();
    }
    else
    {
        return FALSE;
    }
}
else
{
    // Copy the supplied file path.
    strFile = pszFileName;
}

// Try to open the file for write access.
CFile file;
if (!file.Open(strFile, CFile::modeReadWrite | CFile::modeCreate |
CFile::shareExclusive))
{
    AfxMessageBox("Failed to open file");
    return FALSE;
}

BOOL bResult = Save(&file, (GetFileType(strFile) != FILE_BM));
file.Close();
if (!bResult)
    AfxMessageBox("Failed to save file");
return bResult;
}

CDIB::BITMAP_FILE_TYPE CDIB::GetFileType(LPCSTR szFile)
{
    char* p = strrchr(szFile, '.');    // .extension
    if (p)
    {
        p++;
        if (lstrcmpi(p, "BM") == 0)    // BMP, BMZ, BMC, ...
        {
            switch (toupper(p[2]))
            {
                case NULL: return FILE_BM;
                case 'Z': return FILE_BMZ;
                case 'C': return FILE_BMC;
                default: return FILE_BMP; // 'P'
            }
        }
        if (lstrcmpi(p, "GIF") == 0)
            return FILE_GIF;
        if (lstrcmpi(p, "TIF") == 0)
            return FILE_TIF;
        if (lstrcmpi(p, "JPG") == 0)
            return FILE_JPG;
    }
    char szMsg[256];

```

```

        wsprintf(szMsg, "Unrecognized bitmap file type - \"%s\"", szFile);
        AfxMessageBox(szMsg);
        return FILE_NONE;
    }

#ifdef _VICTOR
    //////////////////////////////////////
    //////////////////////////////////////
    // Victor

    // Load a DIB from an open file.
    BOOL CDIB::LoadFromImage(LPVOID pImg)
    {
        BYTE* pSrc = (BYTE*)pImg;
        // Work out how much memory we need for the BITMAPINFO structure, color
        table
        // and then for the bits. Allocate the memory blocks.
        // Always allocate enough room for 256 entries.
        // Allocate the memory for the header.
        BITMAPINFO* pBmpInfo = (LPBITMAPINFO)malloc(sizeof(BITMAPINFOHEADER) +
        256 * sizeof(RGBQUAD));
        if (!pBmpInfo)
        {
            TRACE("Out of memory for DIB header");
            return FALSE;
        }
        // Copy the header
        ::CopyMemory(pBmpInfo, pSrc, sizeof(BITMAPINFOHEADER));
        pSrc += sizeof(BITMAPINFOHEADER);

        int iBitsSize = pBmpInfo->bmiHeader.biSizeImage;
        // Copy the BmpInfoHdr we have so far, and then read in the color table
        from the file.
        int iColorTableSize = NumDIBColorEntries(pBmpInfo) *
        sizeof(RGBQUAD);

        // Read the color table from the file.
        ::CopyMemory(((LPBYTE)pBmpInfo) + sizeof(BITMAPINFOHEADER), pSrc,
        iColorTableSize);
        pSrc += iColorTableSize;

        // Allocate the memory for the bits and read the bits from the file.
        BYTE* pBits = (BYTE*)malloc(iBitsSize);
        if (!pBits)
        {
            TRACE("Out of memory for DIB bits");
            if (pBmpInfo)
                free(pBmpInfo);
            return FALSE;
        }

        // Read the bits.
        ::CopyMemory(pBits, pSrc, iBitsSize);

        // Everything went OK.
        if (m_pBMI)
            free(m_pBMI);
    }

```

```

        m_pBMI = pBmpInfo;
        if (m_bMyBits && m_pBits)
            free(m_pBits);
        m_pBits = pBits;
        m_bMyBits = TRUE;
        return TRUE;
    }

int CDIB::LoadGIF(LPCSTR fname, imgdes* image)
{
    // Get info on the file we're to load
    GifData gdat;
    CString str;
    int rcode = gifinfo((char*)fname, &gdat);
    if (rcode != NO_ERROR) // Fill structure
    {
        str.Format("Error in reading %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Allocate space for an 8-bit image
    rcode = allocimage(image, gdat.width, gdat.length, gdat.vbitcount);
    if (rcode != NO_ERROR)
    {
        str.Format("Not enough memory to load %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Load image
    rcode = loadgif(fname, image);
    if (rcode != NO_ERROR)
    {
        freeimage(image); // Free image on error
        str.Format("Could not load %s", fname);
        AfxMessageBox(str);
    }
    return rcode;
}

int CDIB::LoadTIF(LPCSTR fname, imgdes* image)
{
    // Get info on the file we're to load
    TiffData tdat;
    CString str;
    int rcode = tiffinfo((char*)fname, &tdat);
    if (rcode != NO_ERROR) // Fill structure
    {
        str.Format("Error in reading %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Allocate space for an image
    rcode = allocimage(image, tdat.width, tdat.length, tdat.vbitcount);
    if (rcode != NO_ERROR)
    {
        str.Format("Not enough memory to load %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
}

```

```

    }
    rcode = loadtif(fname, image);
    if (rcode != NO_ERROR) // Free image on error
    {
        freeimage(image);
        str.Format("Could not load %s", fname);
        AfxMessageBox(str);
    }
    return rcode;
}

int CDIB::LoadJPG(LPCSTR fname, imgdes* image)
{
    // Get info on the file we're to load
    JpegData jdat;
    CString str;
    int rcode = jpeginfo(fname, &jdat);
    if (rcode != NO_ERROR) // Fill structure
    {
        str.Format("Error in reading %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Allocate space for an image
    rcode = allocimage(image, (int)jdat.width, (int)jdat.length,
jdat.vbitcount);
    if (rcode != NO_ERROR)
    {
        str.Format("Not enough memory to load %s", fname);
        AfxMessageBox(str);
        return rcode;
    }
    // Load image
    rcode = loadjpg((char*)fname, image);
    if (rcode != NO_ERROR)
    {
        freeimage(image); // Free image on error
        str.Format("Could not load %s", fname);
        AfxMessageBox(str);
    }
    return rcode;
}

// Source image will remain intact.
// Destination image should be a pointer to an imgdes object.
// You should call freeimage(desimg) on success
int CDIB::CreateImage8(imgdes* srcimg, imgdes* desimg)
{
    int rcode;
    //To view the entire image, use a temporary image descriptor
    // and set the image area to the entire image
    imgdes timage;
    copyimgdes(srcimg, &timage);
    setimagearea(&timage, 0, 0,
        (unsigned)timage.bmh->biWidth - 1,
        (unsigned)timage.bmh->biHeight - 1);

```

```

// If we're displaying a 24-bit image on an 8-bit
// display adapter, create a temporary 8-bit image for display
if (srcimg->bmh->biBitCount == 24)
{
    rcode = allocimage(desimg, (int)timage.bmh->biWidth,
(int)timage.bmh->biHeight, 8);
    if (rcode == NO_ERROR)
    {
        // Quick dither representation of 24-bit image
        // (Or use colorsscatter)
        rcode = colordither(&timage, desimg, COLORBITHER256);
        rcode = converttrgbtopal(256, &timage, desimg);
        // If error, free allocated memory
        if (rcode != NO_ERROR)
            freeimage(desimg);
    }
}
else
    copyimgdes(&timage, desimg);
return rcode;
}
#endif // _VICTOR

/*
void CDIB::ReplaceColor(const int nI, const RGBQUAD& rgbQ)
{
    ASSERT(m_pBMI);
    ASSERT(m_pBMI->bmiHeader.biBitCount == 8);
    ASSERT(m_pBits);

    BYTE* pBits = (BYTE*)GetBitsAddress();
    int iSize = StorageWidth() * DibHeight();
    while (iSize--)
    {
        *pBits = imap[*pBits];
        pBits++;
    }
    LPRGBQUAD pctThis = GetClrTabAddress();
    ASSERT(pctThis);
    RGBQUAD oc = pctThis[nWhat];
    pctThis[nWhat] = pctThis[nWith];
    TRACE("CDIB::ReplaceColor (%d,%d,%d)->(%d,%d,%d)\n",
        oc.rgbRed, oc.rgbGreen, oc.rgbBlue,
        pctThis[nWith].rgbRed, pctThis[nWith].rgbGreen,
        pctThis[nWith].rgbBlue);
}
*/

UINT CDIB::SetPaletteEntries(UINT nStartIndex, UINT nNumEntries,
LPPALETTEENTRY lpPaletteColors)
{
    RGBQUAD* pctThis = GetClrTabAddress();
    // ::CopyMemory(pctThis, lpPaletteColors + nStartIndex, nNumEntries *
sizeof(PALETTEENTRY));
    // We can't use CopyMemory function, because the byte order of PALETTEENTRY
(R,G,B,f)
    // is different with that of RGBQUAD (B,G,R,res).

```

```

// RGBQUAD      <- when read from the file (byte order reversed in word)
// PALETTEENTRY <- in memory
    for (UINT i=0; i < nNumEntries; i++)
    {
        int k = nStartIndex + i;
        pctThis[k].rgbRed      = lpPaletteColors[i].peRed;
        pctThis[k].rgbGreen    = lpPaletteColors[i].peGreen;
        pctThis[k].rgbBlue     = lpPaletteColors[i].peBlue;
        pctThis[k].rgbReserved = lpPaletteColors[i].peFlags;
    }
    return 0;
}

/*
UINT CDIB::SetPaletteEntriesFileIndex(UINT nStartIndex, UINT nNumEntries,
LPPALETTEENTRY lpPaletteColors)
{
    RGBQUAD* pctThis = GetClrTabAddress();
    // ::CopyMemory(pctThis, lpPaletteColors + nStartIndex, nNumEntries *
sizeof(PALETTEENTRY));
    // We can't use CopyMemory function, because the byte order of PALETTEENTRY
(R,G,B,f)
    // is different with that of RGBQUAD (B,G,R,res).
    for (UINT i=0; i < nNumEntries; i++)
    {
        int k = m_imap[nStartIndex + i];
        pctThis[k].rgbRed      = lpPaletteColors[i].peRed;
        pctThis[k].rgbGreen    = lpPaletteColors[i].peGreen;
        pctThis[k].rgbBlue     = lpPaletteColors[i].peBlue;
        pctThis[k].rgbReserved = lpPaletteColors[i].peFlags;
    }
    return 0;
}

*/
UINT CDIB::ShiftRGBPercent(UINT nStartIndex, UINT nNumEntries, int nPer)
{
    RGBQUAD* pctThis = GetClrTabAddress();
    if (nPer < -100)
        nPer = -100;
    float fFactor = (float)nPer / 100;
    for (UINT i=0; i < nNumEntries; i++)
    {
        int k = nStartIndex + i;
        int nVal;
        nVal = pctThis[k].rgbRed + (int)(pctThis[k].rgbRed * fFactor);
        if (nVal > 255)
            nVal = 255;
        pctThis[k].rgbRed      = nVal;
        nVal = pctThis[k].rgbGreen + (int)(pctThis[k].rgbGreen *
fFactor);
        if (nVal > 255)
            nVal = 255;
        pctThis[k].rgbGreen    = nVal;
        nVal = pctThis[k].rgbBlue + (int)(pctThis[k].rgbBlue * fFactor);
        if (nVal > 255)
            nVal = 255;
        pctThis[k].rgbBlue     = nVal;
    }
}

```

```

    }
    return 0;
}

UINT CDIB::RotatePaletteIndex(UINT nStartIndex, UINT nNumEntries, UINT nBy)
{
    while (nBy > nNumEntries)
        nBy -= nNumEntries;
    RGBQUAD* pctSt = GetClrTabAddress();
    pctSt += nStartIndex;
    RGBQUAD* pRGBBuf = new RGBQUAD[nNumEntries];
    int nTotalSize = nNumEntries * sizeof(RGBQUAD);
    int nBySize = nBy * sizeof(RGBQUAD);
    ::CopyMemory(pRGBBuf + nBy, pctSt,
        nTotalSize - nBySize);
    ::CopyMemory(pRGBBuf, pctSt + nNumEntries - nBy, nBySize);
    ::CopyMemory(pctSt, pRGBBuf,
        nTotalSize);
    delete [] pRGBBuf;
    return 0;
}
/*
void CDIB::SaveResourceName(const char* pszPath)
{
    if (!pszPath)
        return;
    int nLen = strlen(pszPath);
    char* p = strrchr(pszPath, '\\'); // Exclude path
    if (!p)
        return;
    char* q = strrchr(pszPath, '.'); // Exclude extension
    if (q)
        *q = NULL;
    p++;
    m_strName = (char*)p;
}
*/

```



```

//
//  CDIB:
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//
//  Nov 25mon '96      Added GIF, TIF, JPG support using Victor Library

// #include <mmsystem.h>      // link winmm.lib
#ifdef _VICTOR
#include <vicdefs.h>      // link Vic32ms.lib, runtime Vic32.dll
#endif

#ifndef __DIB_H
#define __DIB_H

typedef struct tagBITMAPCOMBHEADER
{
    WORD  wType;                // 'DS' for Data Source.
    DWORD dwFileSize;          // Total file size.
    WORD  wNumEntries;          // Number of entries
    WORD  bcReserved1;
    WORD  bcReserved2;
} BITMAPCOMBHEADER;

typedef struct tagBMPFILEINFOENTRY
{
    char  id[13];                // org filename (with extension) + NULL
    DWORD dwOffset;              // offset to the BITMAPFILEHEADER
} BMPFILEINFOENTRY;

enum IMAGE_OPERATION
{
    OPACITY_100      = 0x0000,    // 100% default
    OPACITY_75       = 0x0001,
    OPACITY_50       = 0x0002,    // 50%
    OPACITY_25       = 0x0004,    // 25%
    OPACITY_12       = 0x0008,    // 12.5% paint
    OPACITY_0        = 0x0010,    //
    OPACITY_MASK     = 0x001F,
    IMAGE_FLIP       = 0x0100,    // change Mirror state
    IMAGE_VERTICAL   = 0x0200,
    ORIENT_MASK      = 0x0F00,
    USE_COLORKEY     = 0x1000,    // Colorkey
    COLORKEY_MASK    = 0xF000,
    DEFAULT_IO       = 0x1000
};

class CDIB : public CObject
{
public:
    DECLARE_SERIAL(CDIB)
    CDIB();
    // By using virtual destructor, derived classes' destructors can be
    called...

```

```

    virtual ~CDIB();

// Attributes
    BITMAPINFO*      GetBitmapInfoAddress() const { return m_pBMI; } //
Pointer to bitmap info
    void*           GetBitsAddress() const      { return m_pBits; }
// Pointer to the bits
    int             GetBitsSize() const;
    RGBQUAD*        GetClrTabAddress() const      // Pointer
to color table
                                { return (LRGBQUAD) (((BYTE*) (m_pBMI)) +
sizeof(BITMAPINFOHEADER)); }
    int             GetNumClrEntries() const;
// Number of color table entries
    void*           GetPixelAddress(const x, const y) const;
    int             GetBitCount() const      { return m_pBMI->
bmiHeader.biBitCount; }
    virtual int     GetWidth() const { return DibWidth(); } // Image
width
    virtual int     GetHeight() const { return DibHeight(); } // Image
height
    virtual void     GetRect(CRect& rect) const;
    CString*        GetName()              { return &m_strName; }
    enum BITMAP_FILE_TYPE
    {
        FILE_NONE,
        FILE_BMP,
        FILE_BM,      // without palette table
        FILE_BMZ,      // LZ compressed
        FILE_BMC,      // LZ compressed without palette table
        FILE_GIF,
        FILE_TIF,
        FILE_JPG
    };
    BITMAP_FILE_TYPE GetFileType(LPCSTR szFile);

// Operations
    virtual void     Serialize(CArchive& ar);
    BOOL Create(const iWidth, const iHeight, const char*
pszPalFileName=NULL); // Create a new DIB
    BOOL Create(BITMAPINFO* pBMI, BYTE* pBits); // Create
from existing mem
    virtual BOOL     Load(CFile* const fp, const char*
pszPalFileName=NULL); // Load from file
    virtual BOOL     Load(const char* pszFileName=NULL, const char*
pszPalFileName=NULL); // Load DIB from disk file
    BOOL LoadSource(const char* pszSrcFileName=NULL,
const char* pszFileName=NULL,
const char* pszPalFileName=NULL); //
Load DIB from disk file
    virtual BOOL     Load(const WORD wResid); //
Load DIB from resource
    BOOL LoadPalette(const char* pszPalFileName);
    virtual BOOL     Save(const char* pszFileName=NULL); // Save DIB
to disk file
    BOOL SaveSource(const char* pszSrcFileName=NULL,

```

```

const char* pszFileName=NULL);
// Save to Data Source
virtual BOOL Save(CFile* const fp, const BOOL bPalette=TRUE);
// Save to file
virtual void Draw(CDC* pDC, const x, const y);
void Draw(CDC* pDC, const xd, const yd, const w, int h,
const xs, const ys);
void Tile(CDC* pDC, const x0, const y0, CRect& rcClient);
virtual BOOL MapColorsToPalette(const CPalette* pPal);
virtual void CopyBits(CDIB* pDIB,
const xd, const yd, const w, int h,
const xs, const ys, const COLORREF
clrTrans=0xFFFFFFFF,
const WORD wImOp=DEFAULT_IO);
UINT SetPaletteEntries(UINT nStartIndex, UINT nNumEntries,
LPPALETTEENTRY lpPaletteColors);
UINT ShiftRGBPercent(UINT nStartIndex, UINT nNumEntries, int nPer);
UINT RotatePaletteIndex(UINT nStartIndex, UINT nNumEntries, UINT nBy);
void ClearImage();
void ClearRect(CRect& rcClear);

protected:
    BITMAPINFO* m_pBMI; // Pointer to BITMAPINFO struct
    BYTE* m_pBits; // Pointer to the bits
    BOOL m_bMyBits; // TRUE if DIB owns Bits memory
    BOOL m_bMapColorsDone;
    CString m_strName; // BMP Filename, resource name

#ifdef _VICTOR
// For Victor Library
    BOOL LoadFromImage(LPVOID pImg);
    int LoadGIF(LPCSTR fname, imgdes* image);
    int LoadTIF(LPCSTR fname, imgdes* image);
    int LoadJPG(LPCSTR fname, imgdes* image);
    int CreateImage8(imgdes* srcimg, imgdes* desimg);
#endif // _VICTOR

private:
    int DibWidth() const { return m_pBMI->bmiHeader.biWidth; }
    int DibHeight() const { return m_pBMI->bmiHeader.biHeight; }
    int StorageWidth() const { return (m_pBMI->bmiHeader.biWidth
+ 3) & ~3; }
};

/*
// CBmpUsage
class AFX_EXT_CLASS CBmpUsage : public CObject
{
    DECLARE_SERIAL(CBmpUsage)
public:
    CBmpUsage();
    ~CBmpUsage();
    void AddRef(); { ++m_nRef; }
    int Release() { return (--m_nRef); }
}

```

UDSGen\DIB.h

```
protected:
    CString          m_strName;    // Filename
    BITMAPINFO* m_pBMI;            // Pointer to BITMAPINFO struct
    BYTE*          m_pBits;        // Pointer to the bits
    int             m_nRef;         // Reference Count
};

// CBUT : Bitmap Usage Table
class AFX_EXT_CLASS CBUT
{
public:
    CBUT();
    ~CBUT();

};
*/
#endif // __DIB_H
```

```

//
//  CDIBPal :
//
//  (C) Programmed by Kim
//
//  Information Technology Institute
//  UNICHAT INC
//

#include "stdafx.h"
#include "DIBPal.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CDIBPal

CDIBPal::CDIBPal()
{
}

CDIBPal::~CDIBPal()
{
}

// Create a palette from the color table in a DIB.
BOOL CDIBPal::Create(CDIB* pDIB)
{
    DWORD dwColors = pDIB->GetNumClrEntries();
    // Check the DIB has a color table.
    if (!dwColors)
    {
        TRACE("No color table");
        return FALSE;
    }

    // Get a pointer to the RGB quads in the color table.
    RGBQUAD* pRGB = pDIB->GetClrTabAddress();

    // Allocate a log pal and fill it with the color table info.
    LOGPALETTE* pPal = (LOGPALETTE*)malloc(sizeof(LOGPALETTE) + dwColors *
sizeof(PALETTEENTRY));
    if (!pPal)
    {
        TRACE("Out of memory for logpal");
        return FALSE;
    }
    pPal->palVersion = 0x300; // Windows 3.0
    pPal->palNumEntries = (WORD)dwColors; // Table size
    for (DWORD dw=0; dw < dwColors; dw++)
    {
        pPal->palPalEntry[dw].peRed = pRGB[dw].rgbRed;
        pPal->palPalEntry[dw].peGreen = pRGB[dw].rgbGreen;
        pPal->palPalEntry[dw].peBlue = pRGB[dw].rgbBlue;
    }
}

```

```

        pPal->palPalEntry[dw].peFlags = 0;
    }
    BOOL bResult = CreatePalette(pPal);
    free(pPal);
    return bResult;
}

////////////////////////////////////
// CDIBPal commands

int CDIBPal::GetNumColors()
{
    int iColors = 0;
    if (!GetObject(sizeof(iColors), &iColors))
    {
        TRACE("Failed to get num pal colors");
        return 0;
    }
    return iColors;
}

// bBkgnd: Force Background
void CDIBPal::Draw(CDC* pDC, CRect& rect, const BOOL bBkgnd, const BOOL
bShowIndex)
{
    int iColors = GetNumColors();
    CPalette* pOldPal = pDC->SelectPalette(this, bBkgnd);
    pDC->RealizePalette();
    int i, j, top, left, bottom, right;
    for (j=0, top=0; j<16 && iColors; j++, top=bottom)
    {
        bottom = (j+1) * rect.bottom / 16 + 1;
        for (i=0, left=0; i<16 && iColors; i++, left=right)
        {
            right = (i+1) * rect.right / 16 + 1;
            CBrush br(PALETTEINDEX(j * 16 + i));
            CBrush* brold = pDC->SelectObject(&br);
            pDC->Rectangle(left-1, top-1, right, bottom);
            pDC->SelectObject(brold);
            if (bShowIndex)
            {
                CString strIdx;
                strIdx.Format("%0x", j * 16 + i);
                pDC->SetBkMode(TRANSPARENT);
                pDC->SetTextColor(PALETTEINDEX(j * 16 + i));
                //
                Dark Yellow
                pDC->TextOut(left, top, strIdx);
            }
            iColors--;
        }
    }
    pDC->SelectPalette(pOldPal, FALSE);
}

BOOL CDIBPal::SetSysPalColors()
{
    BOOL bResult = FALSE;

```

```

int i, iSysColors, iPalEntries;
HPALETTE hpalOld;

// Get a screen DC to work with.
HWND hwndActive = ::GetActiveWindow();
HDC hdcScreen = ::GetDC(hwndActive);
ASSERT(hdcScreen);

// Make sure we are on a palettized device.
if (! (::GetDeviceCaps(hdcScreen, RASTERCAPS) & RC_PALETTE))
{
    TRACE("Not a palettized device\n");
    goto abort;
}

// Get the number of system colors and the number of palette
// entries. Note that on a palletized device the number of
// colors is the number of guaranteed colors, i.e., the number
// of reserved system colors.
iSysColors = ::GetDeviceCaps(hdcScreen, NUMCOLORS);
iPalEntries = ::GetDeviceCaps(hdcScreen, SIZEPALETTE);

// If there are more than 256 colors we are wasting our time.
if (iSysColors < 0 || iSysColors > 256) goto abort;

// Now we force the palette manager to reset its tables so that
// the next palette to be realized will get its colors in the order
they are
// in the logical palette. This is done by changing the number of
// reserved colors.
::SetSystemPaletteUse(hdcScreen, SYSPAL_NOSTATIC);
::SetSystemPaletteUse(hdcScreen, SYSPAL_STATIC);

// Select our palette into the screen DC and realize it so that
// its colors will be entered into the free slots in the physical
palette.
hpalOld = ::SelectPalette(hdcScreen, (HPALETTE)m_hObject, // Our hpal
FALSE);

#ifdef _DEBUG
    UINT nColorsRemap;
    nColorsRemap = ::RealizePalette(hdcScreen);
    TRACE("CDIBPal::SetSysPalColors() returned %d.\n", nColorsRemap);
#else
    ::RealizePalette(hdcScreen);
#endif

// Now replace the old palette (but don't realize it)
::SelectPalette(hdcScreen, hpalOld, FALSE);

// The physical palette now has our colors set in place and its own
// reserved colors at either end. We can grab the lot now.
PALETTEENTRY pe[256];
GetSystemPaletteEntries(hdcScreen, 0, iPalEntries, pe);

// Set the PC_NOCOLLAPSE flag for each of our colors so that the GDI
// won't merge them. Be careful not to set PC_NOCOLLAPSE for the
// system color entries so that we won't get multiple copies of these
// colors in the palette when we realize it.

```

```

    for (i = 0; i < iSysColors/2; i++)
        pe[i].peFlags = 0;
    for (; i < iPalEntries-iSysColors/2; i++)
        pe[i].peFlags = PC_NOCOLLAPSE;
    for (; i < iPalEntries; i++)
        pe[i].peFlags = 0;

    // Resize the palette in case it was smaller.
    ResizePalette(iPalEntries);

    // Update the palette entries with what is now in the physical palette.
    SetPaletteEntries(0, iPalEntries, pe);
    bResult = TRUE;

abort:
    ::ReleaseDC(hwndActive, hdcScreen);
    return bResult;
}

// Load a palette from a named file.
// To get the PALETTEENTRY only, allocate a memory for it and pass it.
BOOL CDIBPal::Load(const char* pszFileName, PALETTEENTRY* pPE)
{
    CString strFile;

    if ((pszFileName == NULL) || (strlen(pszFileName) == 0))
    {
        // Show an File Open dialog to get the name.
        CFileDialog dlg (TRUE, // Open
                        NULL, // No default extension
                        NULL, // No initial file name
                        OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
                        "Palette files (*.PAL)|*.PAL|All files
(*.*)|*.*||");
        if (dlg.DoModal() == IDOK)
            strFile = dlg.GetPathName();
        else
            return FALSE;
    }
    else
    {
        // Copy the supplied file path.
        strFile = pszFileName;
    }

    // Try to open the file for read access.
    CFile file;
    if (!file.Open(strFile, CFile::modeRead | CFile::shareDenyWrite))
    {
        strFile += " not found!";
        AfxMessageBox(strFile);
        return FALSE;
    }

    BOOL bResult = Load(&file, pPE);
    file.Close();
    if (!bResult)

```



```

        AfxMessageBox("Failed to load palette file");
        return bResult;
    }

    // Load a palette from an open CFile object.
    BOOL CDIBPal::Load(CFile* const fp, PALETTEENTRY* pPE)
    {
        return Load(fp->m_hFile, pPE);
    }

    // Load a palette from an open file handle.
    BOOL CDIBPal::Load(const UINT hFile, PALETTEENTRY* pPE)
    {
        MMIOINFO info;
        memset(&info, 0, sizeof(info));
        info.adwInfo[0] = hFile;
        HMMIO hmmio = mmioOpen(NULL, &info, MMIO_READ | MMIO_ALLOCBUF);
        if (!hmmio)
        {
            TRACE("mmioOpen failed\n");
            return FALSE;
        }
        BOOL bResult = Load(hmmio, pPE);
        mmioClose(hmmio, MMIO_FHOPEN);
        return bResult;
    }

    // Load a palette from an open MMIO handle.
    BOOL CDIBPal::Load(const HMMIO hmmio, PALETTEENTRY* pPE)
    {
        // Check whether it's a RIFF PAL file.
        MMCKINFO ckFile;
        ckFile.fccType = mmioFOURCC('P','A','L',' ');
        if (mmioDescend(hmmio, &ckFile, NULL, MMIO_FINDRIFF) != 0)
        {
            TRACE("Not a RIFF or PAL file\n");
            return FALSE;
        }
        // Find the 'data' chunk.
        MMCKINFO ckChunk;
        ckChunk.ckid = mmioFOURCC('d','a','t','a');
        if (mmioDescend(hmmio, &ckChunk, &ckFile, MMIO_FINDCHUNK) != 0)
        {
            TRACE("No data chunk in file\n");
            return FALSE;
        }
        // Allocate some memory for the data chunk.
        int iSize = ckChunk.cksize;
        void* pdata = malloc(iSize);
        if (!pdata)
        {
            TRACE("No mem for data\n");
            return FALSE;
        }
        // Read the data chunk.
        if (mmioRead(hmmio, (char*)pdata, iSize) != iSize)
        {

```

```

        TRACE("Failed to read data chunk\n");
        free(pdata);
        return FALSE;
    }
    // The data chunk should be a LOGPALETTE structure
    // that we can create a palette from.
    LOGPALETTE* pLogPal = (LOGPALETTE*)pdata;
    if (pLogPal->palVersion != 0x300)
    {
        TRACE("Invalid palette version number\n");
        free(pdata);
        return FALSE;
    }
    // Get the number of entries.
    int iColors = pLogPal->palNumEntries;
    if (iColors <= 0)
    {
        TRACE("No colors in palette\n");
        free(pdata);
        return FALSE;
    }
    // If pPE is specified, do not really create the palette
    if (pPE)
    {
        if (iColors > 256)
        {
            TRACE("Colors in this palette exceeds 256: %dn\n",
iColors);
            iColors = 256;
        }
        ::CopyMemory(pPE, &pLogPal->palPalEntry[0],
iColors*sizeof(PALETTEENTRY));
        free(pdata);
        return TRUE;
    }
    else
    {
        return CreatePalette(pLogPal);
    }
}

// Save a palette to an open CFile object.
BOOL CDIBPal::Save(CFile* const fp)
{
    return Save(fp->m_hFile);
}

// Save a palette to an open file handle.
BOOL CDIBPal::Save(const UINT hFile)
{
    MMIOINFO info;
    memset(&info, 0, sizeof(info));
    info.adwInfo[0] = hFile;
    HMMIO hmmio = mmioOpen(NULL, &info, MMIO_WRITE | MMIO_CREATE |
MMIO_ALLOCBUF);
    if (!hmmio)
    {

```

```

        TRACE("mmioOpen failed\n");
        return FALSE;
    }
    BOOL bResult = Save(hmmio);
    mmioClose(hmmio, MMIO_FHOPEN);
    return bResult;
}

// Save a palette to an open MMIO handle.
BOOL CDIBPal::Save(const HMMIO hmmio)
{
    // Create a RIFF chunk for a PAL file.
    MMCKINFO ckFile;
    ckFile.cksize = 0; // Corrected later
    ckFile.fccType = mmioFOURCC('P','A','L',' ');
    if (mmioCreateChunk(hmmio, &ckFile, MMIO_CREATERIFF) != 0)
    {
        TRACE("Failed to create RIFF-PAL chunk\n");
        return FALSE;
    }
    // Create the LOGPALETTE data which will become the data chunk.
    int iColors = GetNumColors();
    ASSERT(iColors > 0);
    int iSize = sizeof(LOGPALETTE) + (iColors-1) * sizeof(PALETTEENTRY);
    LOGPALETTE* plp = (LOGPALETTE*)malloc(iSize);
    ASSERT(plp);
    plp->palVersion = 0x300;
    plp->palNumEntries = iColors;
    GetPaletteEntries(0, iColors, plp->palPalEntry);
    // create the data chunk.
    MMCKINFO ckData;
    ckData.cksize = iSize;
    ckData.ckid = mmioFOURCC('d','a','t','a');
    if (mmioCreateChunk(hmmio, &ckData, 0) != 0)
    {
        TRACE("Failed to create data chunk\n");
        return FALSE;
    }
    // Write the data chunk.
    if (mmioWrite(hmmio, (char*)plp, iSize) != iSize)
    {
        TRACE("Failed to write data chunk\n");
        free(plp);
        return FALSE;
    }
    free(plp);
    // Ascend from the data chunk which will correct the length.
    mmioAscend(hmmio, &ckData, 0);
    // Ascend from the RIFF-PAL chunk.
    mmioAscend(hmmio, &ckFile, 0);

    return TRUE;
}

```

```

//
//  CDIBPal
//
//  (C) Programmed by Kim,
//
//  Information Technology Institute
//  UNICHAT INC
//

#ifndef __DIBPAL__
#define __DIBPAL__

#include "DIB.h"
#include <mmsystem.h>

class CDIBPal : public CPalette
{
public:
    CDIBPal();
    ~CDIBPal();
    BOOL Create(CDIB* pDIB);          // Create from a DIB
    int  GetNumColors();              // Get the number of colors in the
    palette.
    void Draw(CDC* pDC, CRect& rect, const BOOL bBgnd=FALSE, const BOOL
    bShowIndex=FALSE);
    BOOL SetSysPalColors();
    BOOL Load(const char* pszFileName=NULL, PALETTEENTRY* pPE=NULL);
    BOOL Load(CFile* const fp, PALETTEENTRY* pPE=NULL);
    BOOL Load(const UINT hFile, PALETTEENTRY* pPE=NULL);
    BOOL Load(const HMMIO hmmio, PALETTEENTRY* pPE=NULL);
    BOOL Save(CFile* const fp);
    BOOL Save(const UINT hFile);
    BOOL Save(const HMMIO hmmio);
    int  GetEntries(PALETTEENTRY* pPE);
};

#endif // __DIBPAL__

```

```

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "UDSGen.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    // NOTE - the ClassWizard will add and remove mapping macros
    here.
    // DO NOT EDIT what you see in these blocks of generated code
    !
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))

```

```

    {
        TRACE0("Failed to create toolbar\n");
        return -1;        // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;        // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizable
toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    // be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

////////////////////////////////////
// CMainFrame message handlers

```

```

// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__C3C37D09_AD07_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_MAINFRM_H__C3C37D09_AD07_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Generated message map functions
protected:
   //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    // NOTE - the ClassWizard will add and remove member functions
    here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

```

UDSGen\MainFrm.h

```
#endif //  
!defined(AFX_MAINFRM_H__C3C37D09_AD07_11D1_9169_0000F0610C92__INCLUDED_)
```



```
=====
MICROSOFT FOUNDATION CLASS LIBRARY : UDSGen
=====
```

AppWizard has created this UDSGen application for you. This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that make up your UDSGen application.

UDSGen.h

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CUDSGenApp application class.

UDSGen.cpp

This is the main application source file that contains the application class CUDSGenApp.

UDSGen.rc

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

res\UDSGen.ico

This is an icon file, which is used as the application's icon. This icon is included by the main resource file UDSGen.rc.

res\UDSGen.rc2

This file contains resources that are not edited by Microsoft Developer Studio. You should place all resources not editable by the resource editor in this file.

UDSGen.reg

This is an example .REG file that shows you the kind of registration settings the framework will set for you. You can use this as a .REG file to go along with your application or just delete it and rely on the default RegisterShellFileTypes registration.

UDSGen.clw

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

```
////////////////////////////////////
```

For the main frame window:

MainFrm.h, MainFrm.cpp

These files contain the frame class CMainFrame, which is derived from CFrameWnd and controls all SDI frame features.

res\Toolbar.bmp

This bitmap file is used to create tiled images for the toolbar. The initial toolbar and status bar are constructed in the CMainFrame class. Edit this toolbar bitmap along with the array in MainFrm.cpp to add more toolbar buttons.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

AppWizard creates one document type and one view:

UDSGenDoc.h, UDSGenDoc.cpp - the document

These files contain your CUDSGenDoc class. Edit these files to add your special document data and to implement file saving and loading (via CUDSGenDoc::Serialize).

UDSGenView.h, UDSGenView.cpp - the view of the document

These files contain your CUDSGenView class.

CUDSGenView objects are used to view CUDSGenDoc objects.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Other standard files:

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header (PCH) file named UDSGen.pch and a precompiled types file named StdAfx.obj.

Resource.h

This is the standard header file, which defines new resource IDs. Microsoft Developer Studio reads and updates this file.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you should add to or customize.

If your application uses MFC in a shared DLL, and your application is in a language other than the operating system's current language, you will need to copy the corresponding localized resources MFC40XXX.DLL from the Microsoft Visual C++ CD-ROM onto the system or system32 directory, and rename it to be MFCLOC.DLL. ("XXX" stands for the language abbreviation. For example, MFC40DEU.DLL contains resources translated to German.) If you don't do this, some of the UI elements of your application will remain in the language of the operating system.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by UDSGen.rc
//
#define IDD_ABOUTBOX 100
#define IDD_UDSGEN_FORM 101
#define IDR_MAINFRAME 128
#define IDR_UDSTYPE 129
#define IDD_TARGETNAME 130
#define IDC_LIST1 1000
#define IDC_TARGET 1001
#define IDC_PROGRESS1 1002
#define IDC_GENERATE 1003
#define IDC_QUIT 1004
#define IDC_EDIT1 1005

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS 1
#define _APS_NEXT_RESOURCE_VALUE 131
#define _APS_NEXT_COMMAND_VALUE 32771
#define _APS_NEXT_CONTROL_VALUE 1006
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif

```

UDSGen\StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//     UDSGen.pch will be the pre-compiled header
//     stdafx.obj will contain the pre-compiled type information
```

```
#include "stdafx.h"
```

```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H_C3C37D07_AD07_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_STDAFX_H_C3C37D07_AD07_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>             // MFC core and standard components
#include <afxext.h>             // MFC extensions
#include <afxdisp.h>           // MFC OLE automation classes
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H_C3C37D07_AD07_11D1_9169_0000F0610C92__INCLUDED_)

```

```
// TargetDlg.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "UDSGen.h"
#include "TargetDlg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CTargetDlg dialog
```

```
CTargetDlg::CTargetDlg(CWnd* pParent /*=NULL*/)
: CDialog(CTargetDlg::IDD, pParent)
```

```
{
   //{{AFX_DATA_INIT(CTargetDlg)
    m_strTargetFile = _T("");
   //}}AFX_DATA_INIT
}
```

```
void CTargetDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTargetDlg)
    DDX_Control(pDX, IDC_EDIT1, m_ctlTarget);
    DDX_Text(pDX, IDC_EDIT1, m_strTargetFile);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CTargetDlg, CDialog)
```

```
    //{{AFX_MSG_MAP(CTargetDlg)
```

```
    ON_WM_CREATE()
```

```
    //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CTargetDlg message handlers
```

```
int CTargetDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
    if (CDialog::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO: Add your specialized creation code here
    m_ctlTarget.SetFocus();
    return 0;
}
```

```

#if
!defined(AFX_TARGETDLG_H__C3C37D1B_AD07_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_TARGETDLG_H__C3C37D1B_AD07_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// TargetDlg.h : header file
//

/////////////////////////////////////////////////////////////////
// CTargetDlg dialog

class CTargetDlg : public CDialog
{
// Construction
public:
    CTargetDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(CTargetDlg)
    enum { IDD = IDD_TARGETNAME };
    CEdit m_ctlTarget;
    CString m_strTargetFile;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTargetDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CTargetDlg)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_TARGETDLG_H__C3C37D1B_AD07_11D1_9169_0000F0610C92__INCLUDED_)

```

```
// UDSGen.cpp : Defines the class behaviors for the application.
//
```

```
#include "stdafx.h"
#include "UDSGen.h"
```

```
#include "MainFrm.h"
#include "UDSGenDoc.h"
#include "UDSGenView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CUDSGenApp
```

```
BEGIN_MESSAGE_MAP(CUDSGenApp, CWinApp)
    //{AFX_MSG_MAP(CUDSGenApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // NOTE - the ClassWizard will add and remove mapping macros
here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //{AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CUDSGenApp construction
```

```
CUDSGenApp::CUDSGenApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
```

```
////////////////////////////////////
// The one and only CUDSGenApp object
```

```
CUDSGenApp theApp;
```

```
////////////////////////////////////
// CUDSGenApp initialization
```

```
BOOL CUDSGenApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
```



```

    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a
shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC
statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings(); // Load standard INI file options (including
MRU)

    // Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CUDSGenDoc),
        RUNTIME_CLASS(CMainFrame),           // main SDI frame window
        RUNTIME_CLASS(CUDSGenView));
    AddDocTemplate(pDocTemplate);

    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();

    return TRUE;
}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:

```

```

    CAboutDlg();

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
    // No message handlers
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CUDSGenApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CUDSGenApp commands

```

```

// UDSGen.h : main header file for the UDSGEN application
//

#if !defined(AFX_UDSGEN_H__C3C37D05_AD07_11D1_9169_0000F0610C92__INCLUDED_)
#define AFX_UDSGEN_H__C3C37D05_AD07_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"          // main symbols

////////////////////////////////////
// CUDSGenApp:
// See UDSGen.cpp for the implementation of this class
//

class CUDSGenApp : public CWinApp
{
public:
    CUDSGenApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUDSGenApp)
public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CUDSGenApp)
    afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions
    here.
    // DO NOT EDIT what you see in these blocks of generated code
    !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_UDSGEN_H__C3C37D05_AD07_11D1_9169_0000F0610C92__INCLUDED_)

```

```

// UDSGenDoc.cpp : implementation of the CUDSGenDoc class
//

#include "stdafx.h"
#include "UDSGen.h"

#include "UDSGenDoc.h"
#include "UDSGenView.h"

#include "dib.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CUDSGenDoc

IMPLEMENT_DYNCREATE(CUDSGenDoc, CDocument)

BEGIN_MESSAGE_MAP(CUDSGenDoc, CDocument)
    //{{AFX_MSG_MAP(CUDSGenDoc)
    ON_COMMAND(ID_FILE_NEW, OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CUDSGenDoc construction/destruction

CUDSGenDoc::CUDSGenDoc()
{
    // TODO: add one-time construction code here

    m_dsHeader.wType = 0x5344;
    m_dsHeader.dwFileSize = 0;
    m_dsHeader.wNumEntries = 0;
    m_dsHeader.bcReserved1 = 0;
    m_dsHeader.bcReserved2 = 0;

    m_aDSEntry = NULL;

    //m_targetSize = 0; // Target file size.

    m_pFilePointers = NULL;
}

CUDSGenDoc::~CUDSGenDoc()
{
    if(m_aDSEntry)
        delete m_aDSEntry;

    if(m_pFilePointers)
        delete m_pFilePointers;
}

```

```

BOOL CUDSGenDoc::OnNewDocument()
{

```

```

    if (!CDocument::OnNewDocument())
        return FALSE;

```

```

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

```

```

    return TRUE;
}

```

```

/////////////////////////////////////////////////////////////////
// CUDSGenDoc serialization

```

```

void CUDSGenDoc::Serialize(CArchive& ar)
{

```

```

    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

```

```

/////////////////////////////////////////////////////////////////
// CUDSGenDoc diagnostics

```

```

#ifdef _DEBUG
void CUDSGenDoc::AssertValid() const
{
    CDocument::AssertValid();
}

```

```

void CUDSGenDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

```

```

/////////////////////////////////////////////////////////////////
// CUDSGenDoc commands

```

```

CUDSGenView* CUDSGenDoc::GetView()
{
    POSITION pos;
    pos = GetFirstViewPosition();
    ASSERT(pos);
    CUDSGenView* pView = (CUDSGenView*)GetNextView(pos);
    ASSERT(pView);
    ASSERT(pView->IsKindOf(RUNTIME_CLASS(CUDSGenView)));
}

```

```

    return pView;
}

void CUDSGenDoc::OnFileNew()
{
    // TODO: Add your command handler code here
    GetView()->m_lstSource.ResetContent();

    CFileDialog ofd(TRUE, NULL, NULL,
        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT | OFN_ALLOWMULTISELECT ,
        "Bitmap files|*.DIB;*.BMP;*.BM;*.gif;*.jpg;*.tif|All files
(*.*)|*.*||");

    //CString buffer;
    TCHAR buffer[4090*2];
    memset(buffer,0,4090*2);

    //ofd.m_ofn.lpstrFile = (LPSTR)buffer.GetBuffer(4090);
    ofd.m_ofn.lpstrFile = buffer;
    ofd.m_ofn.nMaxFile = 4090*2;
    if(ofd.DoModal() == IDOK)
    {
        POSITION pos = ofd.GetStartPosition();
        while(pos)
        {
            CString file = ofd.GetNextPathName(pos);
            GetView()->m_lstSource.AddString(file);
        }

        m_nDSEntries = GetView()->m_lstSource.GetCount();
        m_dsHeader.wNumEntries = m_nDSEntries; // How many Files?
        m_aDSEntry = new DATASOURCEENTRY[m_nDSEntries];
        m_pFilePointers = new DWORD[m_nDSEntries];
    }
}

BOOL CUDSGenDoc::Generate(CString fileName)
{
    TRACE("CUDSGenDoc::Generate\n");

    GetView()->SetFilename(fileName);

    CFile target;
    CFileException fileException;
    if ( !target.Open(fileName, CFile::modeCreate | CFile::modeWrite,
&fileException))
    {
        TRACE( "Can't open file %s, error = %u\n", fileName,
fileException.m_cause );
    }

    //target.SetLength(fileLength); // total target file size set.

    //example for CFile::Seek
    target.SeekToBegin(); // Move file pointer to beginning point.

```

```

    if(!SaveHeader(&target))
    {
        AfxMessageBox("Write header error");
    }

    m_dwEntryStartPoint = target.GetPosition();

    int offset = (sizeof(DATASOURCEENTRY))*m_nDSEntries;
    target.Seek(offset, CFile::current);

    for(int j=0 ; j<m_nDSEntries; j++)
    {
        //SaveWithCDIB(&target, j); // no palette.
        SaveContent(&target, j); // Rawdata to files.
    }

    int offsetentry = sizeof(DATASOURCEHEADER);
    target.Seek(offsetentry, CFile::begin);

    for(int i=0 ; i<m_nDSEntries; i++)
    {
        SaveEntry(&target, i);
        //SaveEntryBMP(&target, i);
    }

    GeneratedDSI(fileName); // data source information file.
    return TRUE;
}

BOOL CUDSGenDoc::SaveHeader(CFile * fp)
{
    TRACE("CUDSGenDoc::SaveHeader\n");

    DATASOURCEHEADER dsh;

    // Construct the file header.
    dsh.wType = 0x5344; // 'DS'
    dsh.dwFileSize = GetFileSize();
    dsh.wNumEntries = m_nDSEntries;
    dsh.bcReserved1 = 0;
    dsh.bcReserved2 = 0;

    // Write the file header.
    WORD iSize = sizeof(dsh);
    TRY
    {
        fp->Write(&dsh, iSize);
    }
    CATCH(CFileException, e)
    {
        TRACE("Failed to write file header");
        return FALSE;
    } END_CATCH

    return TRUE;
}

```

```

DWORD CUDSGenDoc::GetFileSize()
{
    TRACE("CUDSGenDoc::GetFileSize\n");

    DWORD fileLength;
    fileLength = 0;
    for(int i=0; i<m_nDSEntries; i++)
    {
        CString tmpfile;
        GetView()->m_lstSource.GetText(i, tmpfile);
        CFile temp(tmpfile, CFile::modeRead);
        fileLength += temp.GetLength(); // size of all file selected.
        temp.Close();
    }

    fileLength += sizeof(DATASOURCEHEADER);
    fileLength += m_nDSEntries * sizeof(DATASOURCEENTRY);

    return fileLength;
}

BOOL CUDSGenDoc::SaveEntry(CFile * fp, int index)
{
    TRACE("CUDSGenDoc::SaveEntry\n");

    DATASOURCEENTRY dse;
    DWORD fpos = 0;
    CString tmpfile;
    GetView()->m_lstSource.GetText(index, tmpfile);

    CFile temp(tmpfile, CFile::modeRead);
    // Construct the file header.
    CString strFN(temp.GetFileName());
    //strFN += ".bmp"; // What kind of file?
    strFN.MakeLower();

    //ASSERT(strFN.GetLength() <= 12);
    ::ZeroMemory(&dse, sizeof(dse));

    lstrcpy(dse.id, strFN);

    /*
    for(int i=0; i<index; i++)
    {
        CString tmpfile;
        GetView()->m_lstSource.GetText(i, tmpfile);
        CFile temp(tmpfile, CFile::modeRead);
        fpos += temp.GetLength(); // size of all file selected.
        temp.Close();
    }

    DWORD off = sizeof(DATASOURCEHEADER) +
        sizeof(DATASOURCEENTRY)*m_nDSEntries + fpos;
    */
    dse.dwOffset = m_pFilePointers[index];

    // Write the file header.

```



```

    int iSize = sizeof(DATASOURCEENTRY);
    TRACE("sizeof(dse)=%d\n", iSize);
    TRY
    {
        fp->Write(&dse, iSize);
        return TRUE;
    }
    CATCH(CFileException, e)
    {
        TRACE("Failed to write file header");
        return FALSE;
    } END_CATCH
}

BOOL CUDSGenDoc::SaveEntryNoPal(CFile * fp, int index)
{
    TRACE("CUDSGenDoc::SaveEntry\n");

    DATASOURCEENTRY dse;
    DWORD fpos = 0;
    CString tmpfile;
    GetView()->m_lstSource.GetText(index, tmpfile);

    CFile temp(tmpfile, CFile::modeRead);
    // Construct the file header.
    CString strFN(temp.GetFileName());
    //strFN += ".bmp"; // What kind of file?
    strFN.MakeLower();

    //ASSERT(strFN.GetLength() <= 12);
    ::ZeroMemory(&dse, sizeof(dse));

    lstrcpy(dse.id, strFN);

    /*
    for(int i=0; i<index; i++)
    {
        CString tmpfile;
        GetView()->m_lstSource.GetText(i, tmpfile);
        CFile temp(tmpfile, CFile::modeRead);
        fpos += temp.GetLength(); // size of all file selected.
        temp.Close();
    }

    DWORD off = sizeof(DATASOURCEHEADER) +
        sizeof(DATASOURCEENTRY)*m_nDSEntries + fpos;
    */
    dse.dwOffset = m_pFilePointers[index];

    // Write the file header.
    int iSize = sizeof(DATASOURCEENTRY);
    TRACE("sizeof(dse)=%d\n", iSize);
    TRY
    {
        fp->Write(&dse, iSize);
        return TRUE;
    }

```

```

    }
    CATCH(CFileException, e)
    {
        TRACE("Failed to write file header");
        return FALSE;
    } END_CATCH

}

BOOL CUDSGenDoc::SaveContent(CFile * fp, int index)
{
    TRACE("CUDSGenDoc::SaveContent\n");
    // Any kind of files to UDS. just binding.

    m_pFilePointers[index] = fp->GetPosition();

    CString tmpfile;
    GetView()->m_lstSource.GetText(index, tmpfile);

    CFile temp(tmpfile, CFile::modeRead);
    int nSize = temp.GetLength();
    BYTE* buffer = (BYTE*)malloc(nSize);
    temp.Read(buffer, nSize);

    fp->Write(buffer, nSize);

    free(buffer);
    return TRUE;
}

void CUDSGenDoc::GenerateDSI(CString fileName)
{
    int l = fileName.GetLength();
    CString dsi = fileName.Left(l-4);
    dsi += ".dsi";
    CFile fnDSI;
    CFileException fileException;
    if ( !fnDSI.Open(dsi, CFile::modeCreate | CFile::modeWrite,
&fileException))
    {
        TRACE( "Can't open file %s, error = %u\n", fileName,
fileException.m_cause );
    }
    for(int i=0; i<m_nDSEntries; i++)
    {
        //char line[13];
        CString tmpfile;
        GetView()->m_lstSource.GetText(i, tmpfile);
        CFile temp(tmpfile, CFile::modeRead);

        CString strFN(temp.GetFileName());
        strFN.MakeLower();
        //lstrcpy(dse.id, strFN);
        strFN += "\r\n";
        fnDSI.Write(strFN, strFN.GetLength());
        temp.Close();
    }
}

```

```

    }
}

void CUDSGenDoc::OnFileOpen()
{
    // TODO: Add your command handler code here
    //GetView()->m_lstSource.ResetContent();

    CFileDialog ofd(TRUE, NULL, NULL,
        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
        "DSI files|*.dsi|*.*||");
    if(ofd.DoModal() == IDOK)
    {
        CStdioFile dsiFile(ofd.GetFileName(), CFile::modeRead);
        CString strTemp;
        while(dsiFile.ReadString(strTemp))
        {
            GetView()->m_lstSource.AddString(strTemp);
        }
    }
}

BOOL CUDSGenDoc::SaveContentNoPal(CFile * fp, int index)
{
    TRACE("CUDSGenDoc::SaveContentNoPal\n");

    m_pFilePointers[index] = fp->GetPosition();

    CString tmpfile;
    GetView()->m_lstSource.GetText(index, tmpfile);

    CDIB* dib = new CDIB;
    dib->Load(tmpfile);

    dib->Save(fp, FALSE); // no palette.

    delete dib;

    return TRUE;
}

BOOL CUDSGenDoc::SaveWithCDIB(CFile * fp, int index)
{
    TRACE("CUDSGenDoc::SaveWithCDIB\n");

    m_pFilePointers[index] = fp->GetPosition();

    CString tmpfile;
    GetView()->m_lstSource.GetText(index, tmpfile);

    CDIB* dib = new CDIB;
    dib->Load(tmpfile);

    dib->Save(fp, FALSE); // no palette.
}

```

```

        delete dib;

    return TRUE;
}

DWORD CUDSGenDoc::GetFileSizeNoPal()
{
    TRACE("CUDSGenDoc::GetFileSize\n");

    DWORD fileLength;
    fileLength = 0;
    for(int i=0; i<m_nDSEntries; i++)
    {
        CString tmpfile;
        GetView()->m_lstSource.GetText(i, tmpfile);
        CFile temp(tmpfile, CFile::modeRead);
        fileLength += temp.GetLength(); // size of all file selected.
        temp.Close();
    }

    fileLength += sizeof(DATASOURCEHEADER);
    fileLength += m_nDSEntries * sizeof(DATASOURCEENTRY);
    fileLength -= m_nDSEntries * (sizeof(RGBQUAD)*255);

    return fileLength;
}

BOOL CUDSGenDoc::SaveHeaderNoPal(CFile * fp)
{
    TRACE("CUDSGenDoc::SaveHeader\n");

    DATASOURCEHEADER dsh;

    // Construct the file header.
    dsh.wType = 0x5344; // 'DS'
    dsh.dwFileSize = GetFileSizeNoPal();
    dsh.wNumEntries = m_nDSEntries;
    dsh.bcReserved1 = 0;
    dsh.bcReserved2 = 0;

    // Write the file header.
    WORD iSize = sizeof(dsh);
    TRY
    {
        fp->Write(&dsh, iSize);
    }
    CATCH(CFileException, e)
    {
        TRACE("Failed to write file header");
        return FALSE;
    } END_CATCH

    return TRUE;
}

```

```

}

BOOL CUDSGenDoc::GenerateNoPal(CString fileName)
{
    TRACE("CUDSGenDoc::GenerateNoPal\n");

    GetView()->SetFilename(fileName);

    CFile target;
    CFileException fileException;
    if ( !target.Open(fileName, CFile::modeCreate | CFile::modeWrite,
&fileException))
    {
        TRACE( "Can't open file %s, error = %u\n", fileName,
fileException.m_cause );
    }

    //target.SetLength(fileLength); // total target file size set.

    //example for CFile::Seek
    target.SeekToBegin(); // Move file pointer to beginning point.

    if(!SaveHeaderNoPal(&target))
    {
        AfxMessageBox("Write header error");
    }

    m_dwEntryStartPoint = target.GetPosition();

    int offset = (sizeof(DATASOURCEENTRY))*m_nDSEntries;
    target.Seek(offset, CFile::current);

    for(int j=0 ; j<m_nDSEntries; j++)
    {
        SaveContentNoPal(&target, j);
    }

    int offsetentry = sizeof(DATASOURCEHEADER);
    target.Seek(offsetentry, CFile::begin);

    for(int i=0 ; i<m_nDSEntries; i++)
    {
        SaveEntryNoPal(&target, i);
        //SaveEntryBMP(&target, i);
    }

    GeneratedDSI(fileName); // data source information file.
    return TRUE;
}

```

```

// UDSGenDoc.h : interface of the CUDSGenDoc class
//
/////////////////////////////////////////////////////////////////

#ifdef AFX_UDSGENDOC_H__C3C37D0B_AD07_11D1_9169_0000F0610C92__INCLUDED_
#define AFX_UDSGENDOC_H__C3C37D0B_AD07_11D1_9169_0000F0610C92__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

typedef struct tagDATASOURCEHEADER
{
    WORD wType; // 'DS' for Data Source.
    DWORD dwFileSize; // Total file size.
    WORD wNumEntries; // Number of entries
    WORD bcReserved1;
    WORD bcReserved2;
} DATASOURCEHEADER;

typedef struct tagDATASOURCEENTRY
{
    char id[13]; // org filename (with extension) + NULL // 13
    bytes.
    DWORD dwOffset; // offset to the BITMAPFILEHEADER // 4 bytes.
} DATASOURCEENTRY, FAR *LPDATASOURCEENTRY; // total 17 bytes.

class CUDSGenView;

class CUDSGenDoc : public CDocument
{
protected: // create from serialization only
    CUDSGenDoc();
    DECLARE_DYNCREATE(CUDSGenDoc)

// Attributes
public:
    DATASOURCEHEADER* GetHeader() { return &m_dsHeader;}
    LPDATASOURCEENTRY GetEntry() { return m_aDSEntry;}

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUDSGenDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

// Implementation
public:
    BOOL GenerateNoPal(CString fileName);
    BOOL Generate(CString fileName);
    virtual ~CUDSGenDoc();

```

```

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    BOOL SaveHeaderNoPal(CFile* fp);
    DWORD GetFileSizeNoPal();
    DWORD* m_pFilePointers;
    DWORD m_dwEntryStartPoint;
    BOOL SaveWithCDIB(CFile* fp, int index);
    void GenerateDSI(CString fileName);
    BOOL SaveContentNoPal(CFile* fp, int index);
    BOOL SaveContent(CFile* fp, int index);
    BOOL SaveEntry(CFile* fp, int index);
    BOOL SaveEntryNoPal(CFile* fp, int index);
    DWORD GetFileSize();
    BOOL SaveHeader(CFile* fp);
    //DWORD m_targetSize;
    CUDSGenView* GetView();
    LPDATASOURCEENTRY m_aDSEntry;
    int m_nDSEntries;
    DATASOURCEHEADER m_dsHeader;
    //{AFX_MSG(CUDSGenDoc)
    afx_msg void OnFileNew();
    afx_msg void OnFileOpen();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_UDSGENDOC_H__C3C37D0B_AD07_11D1_9169_0000F0610C92__INCLUDED_)

```

```

// UDSGenView.cpp : implementation of the CUDSGenView class
//

#include "stdafx.h"
#include "UDSGen.h"

#include "UDSGenDoc.h"
#include "UDSGenView.h"
#include "TargetDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CUDSGenView

IMPLEMENT_DYNCREATE(CUDSGenView, CFormView)

BEGIN_MESSAGE_MAP(CUDSGenView, CFormView)
    //{AFX_MSG_MAP(CUDSGenView)
    ON_BN_CLICKED(IDC_GENERATE, OnGenerate)
    //}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////
// CUDSGenView construction/destruction

CUDSGenView::CUDSGenView()
    : CFormView(CUDSGenView::IDD)
{
    //{AFX_DATA_INIT(CUDSGenView)
    // NOTE: the ClassWizard will add member initialization here
    //}AFX_DATA_INIT
    // TODO: add construction code here
}

CUDSGenView::~CUDSGenView()
{
}

void CUDSGenView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CUDSGenView)
    DDX_Control(pDX, IDC_PROGRESS1, m_ctlProgress);
    DDX_Control(pDX, IDC_LIST1, m_lstSource);
    //}AFX_DATA_MAP
}

```



```

BOOL CUDSGenView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CFormView::PreCreateWindow(cs);
}

////////////////////////////////////
// CUDSGenView printing

BOOL CUDSGenView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CUDSGenView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CUDSGenView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

void CUDSGenView::OnPrint(CDC* pDC, CPrintInfo*)
{
    // TODO: add code to print the controls
}

////////////////////////////////////
// CUDSGenView diagnostics

#ifdef _DEBUG
void CUDSGenView::AssertValid() const
{
    CFormView::AssertValid();
}

void CUDSGenView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CUDSGenDoc* CUDSGenView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CUDSGenDoc)));
    return (CUDSGenDoc*)m_pDocument;
}
#endif // _DEBUG

////////////////////////////////////
// CUDSGenView message handlers

void CUDSGenView::OnGenerate()

```

UDSGen\UDSGenView

```
{
    TRACE("CUDSGenView::OnGenerate\n");

    // TODO: Add your control notification handler code here
    CTargetDlg dlg;
    if(dlg.DoModal() == IDOK)
    {
        CString fn = dlg.m_strTargetFile;
        fn += ".uds";
        GetDocument()->Generate(fn);
    }
}

void CUDSGenView::SetFilename(CString fileName)
{
    CStatic* title = (CStatic*)GetDlgItem(IDC_TARGET);
    title->SetWindowText(fileName);
}
}
```

UDSGen\UDSGenView

```
// UDSGenView.h : interface of the CUDSGenView class
//
/////////////////////////////////////////////////////////////////

#ifdef _MSC_VER
#define AFX_UDSGENVIEW_H__C3C37D0D_AD07_11D1_9169_0000F0610C92__INCLUDED_
#endif

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CUDSGenView : public CFormView
{
protected: // create from serialization only
    CUDSGenView();
    DECLARE_DYNCREATE(CUDSGenView)

public:
    //{{AFX_DATA(CUDSGenView)
    enum { IDD = IDD_UDSGEN_FORM };
    CProgressCtrl      m_ctlProgress;
    CListBox           m_lstSource;
    //}}AFX_DATA

    // Attributes
public:
    CUDSGenDoc* GetDocument();

    // Operations
public:

    // Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CUDSGenView)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL

    // Implementation
public:
    void SetFilename(CString fileName);
    virtual ~CUDSGenView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

    // Generated message map functions

```

UDSGen\UDSGenView.h

protected:

```
//{{AFX_MSG(CUDSGenView)
afx_msg void OnGenerate();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
```

};

```
#ifndef _DEBUG // debug version in UDSGenView.cpp
inline CUDSGenDoc* CUDSGenView::GetDocument()
{ return (CUDSGenDoc*)m_pDocument; }
#endif
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.
```

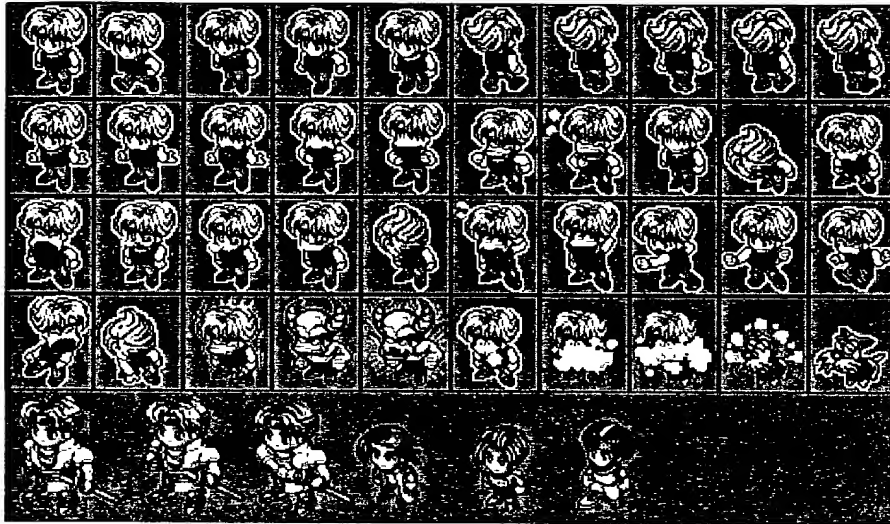
```
#endif //
!defined(AFX_UDSGENVIEW_H__C3C37D0D_AD07_11D1_9169_0000F0610C92__INCLUDED_)
```

sourceuni

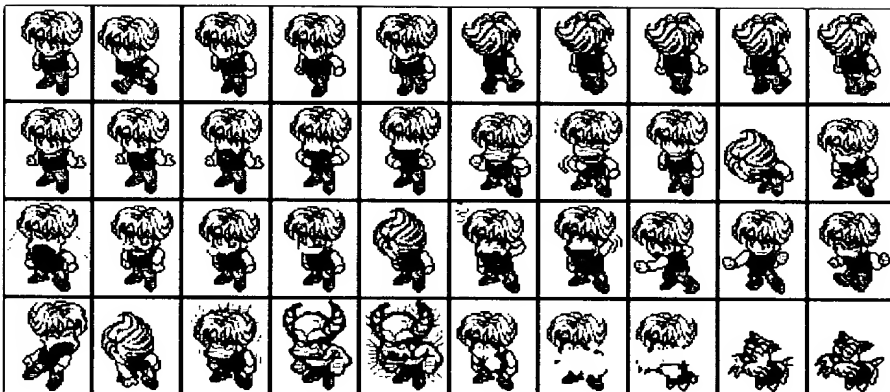
sourceuni.man00.bmp



sourceuni.Man-test.bmp

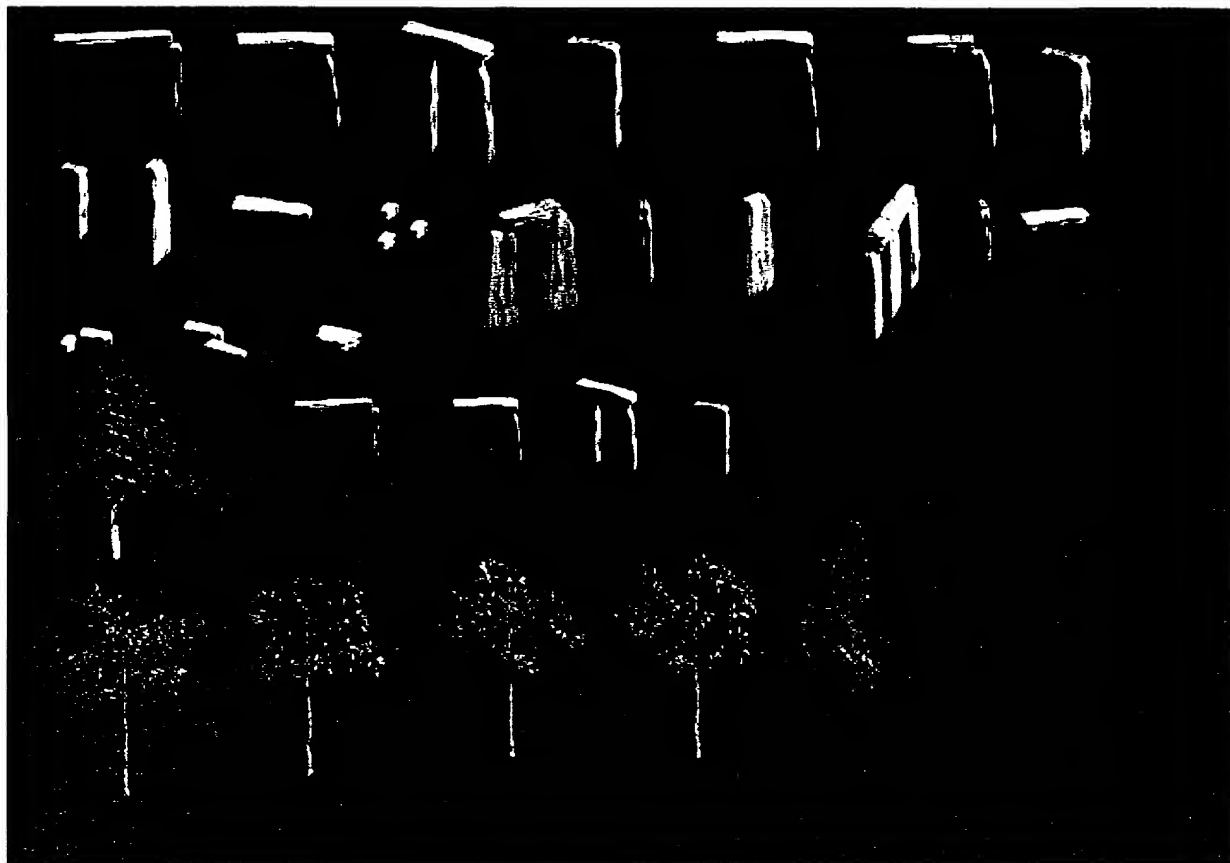


sourceuni.Man00.bmp

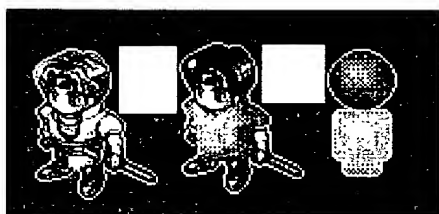


sourceuni

sourceuni.sprite000.bmp

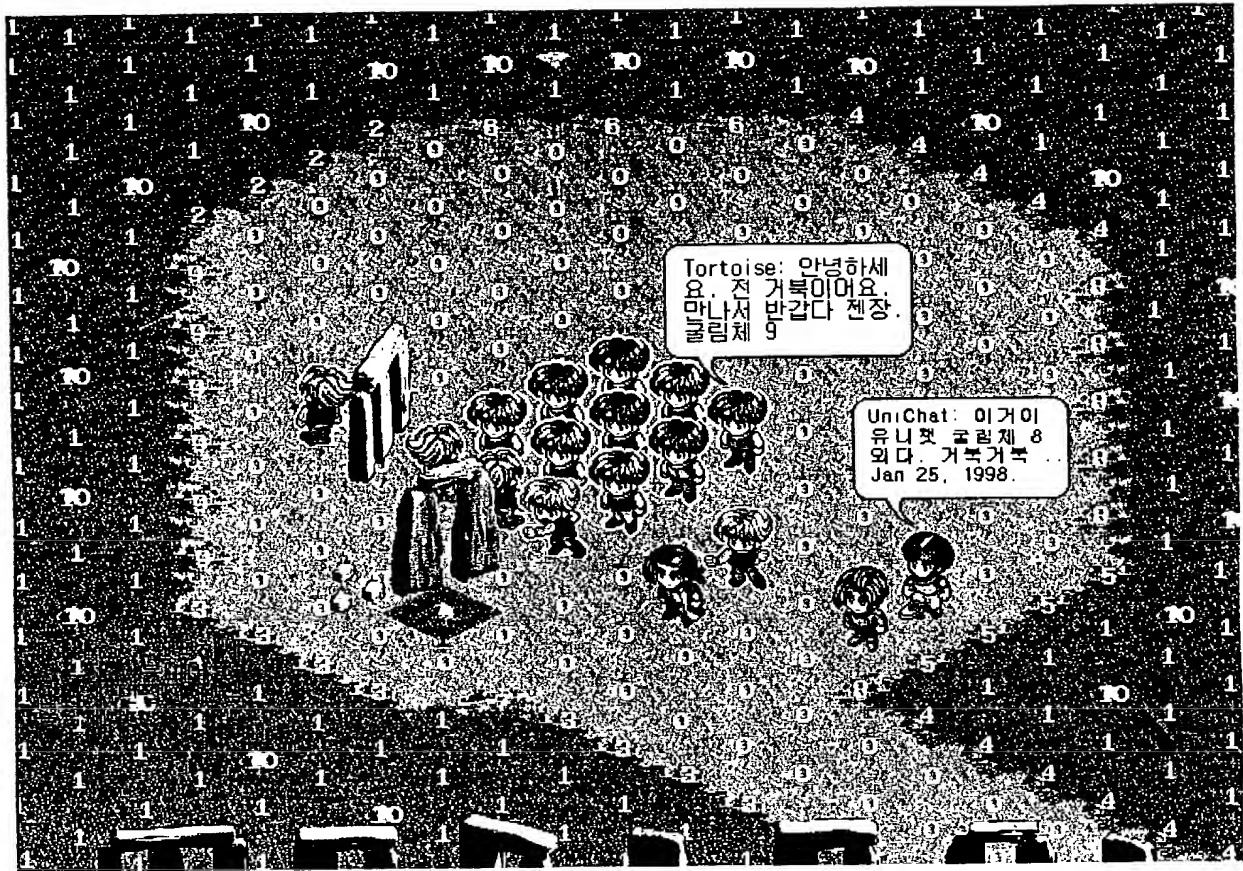


sourceuni.test-uni.bmp



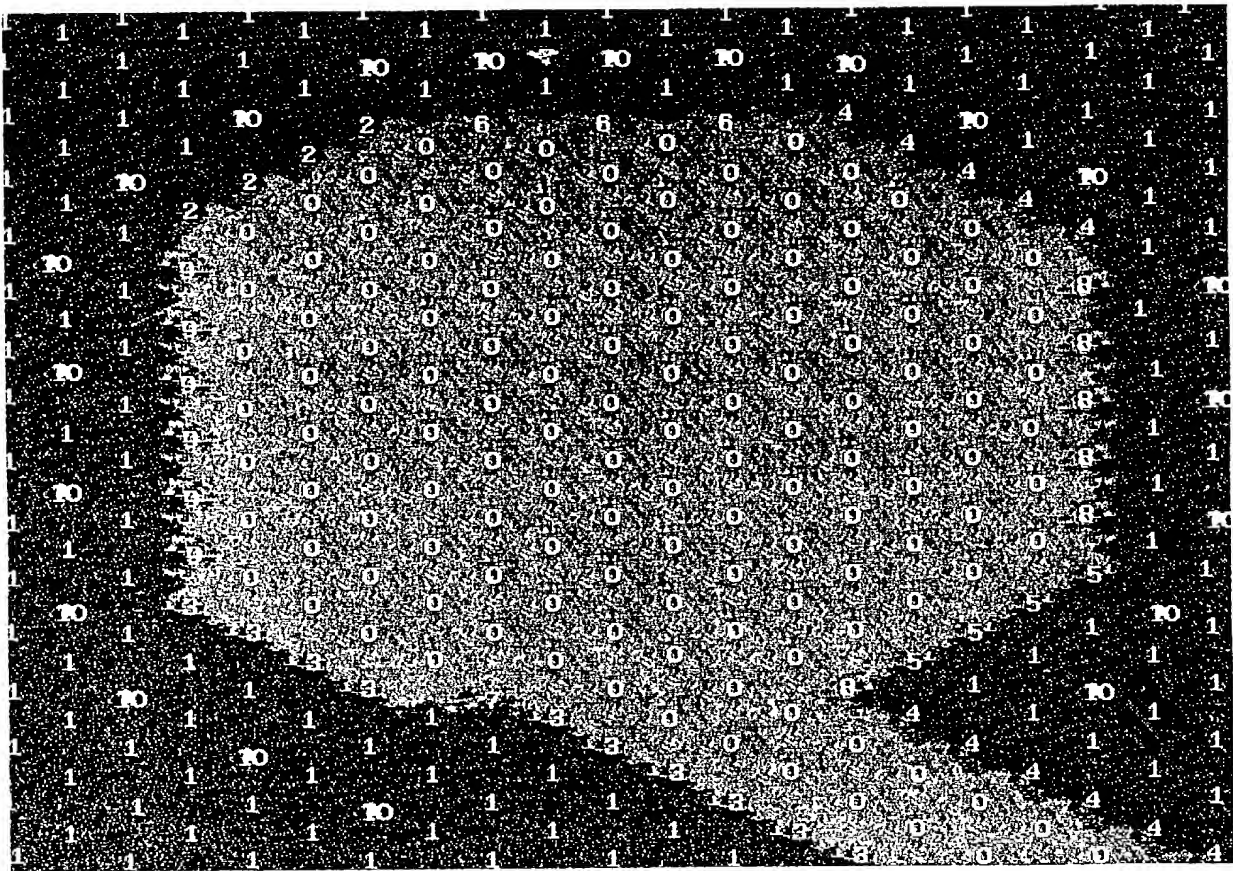
sourceuni

sourceuni.test0.bmp



sourceuni

sourceuni.testmap.bmp



MonTool

MonTool.away.bmp



MonTool.chatpriv.bmp



MonTool.host.bmp



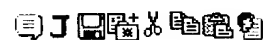
MonTool.msn.bmp



MonTool.sendbar.bmp



MonTool.toolbar.bmp



MonTool.chat.bmp



MonTool.Folder.bmp



MonTool.member.bmp



MonTool.part.bmp



MonTool.spec.bmp

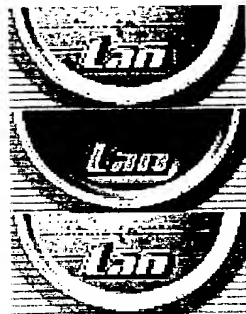


MonTool.world.bmp



USDGen

USDGen.MbtnLan.bmp



UDSGen.MBtNmDm.bmp



USDGen.MbtnNo.bmp

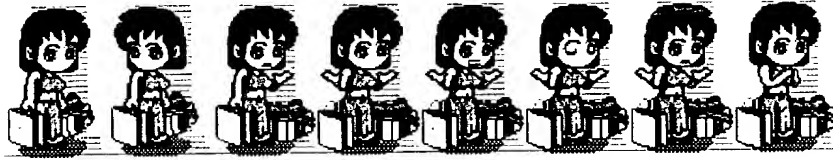


USDGen.MLgnOk.bmp



USDGen

USDGen.MLgnAni.bmp



USDGen.MLoginBk.bmp



[illegible]

KYUNAM KIM

M-8786 US

Unichat Architectural Specification



| | |
|--------------------------------------------------------|-----------|
| 1. SOURCE FILES | 2 |
| PROJECT UC2ANI WHICH GENERATES UC2ANI.DLL | 3 |
| PROJECT MAPED WHICH GENERATES MAPED.EXE | 3 |
| PROJECT UC2 WHICH GENERATES UC2.EXE | 4 |
| 2. DATA FILES | 5 |
| RIT – RESOURCE INFORMATION TABLE | 5 |
| SIT – STAGE INFORMATION TABLE | 9 |
| UDS – UNICHAT DATA SOURCE | 10 |
| 3. DATA MODEL | 11 |
| ANIMATION MECHANISM | 12 |
| MEMORY MODEL FOR THE STAGE AND ACTORS | 14 |
| RESOURCE MANAGER AND ACTOR DESCRIPTION TABLES | 15 |
| 4. COMMUNICATION ARCHITECTURE USING MS CHATSOCK | 16 |
| MS CHATSOCK API | 16 |
| INTEGRATION WITH MFC CLASSES | 17 |
| ACTION COMMANDS | 18 |
| 5. TILE THEORIES | 20 |
| TILE ATTRIBUTES | 20 |
| TILE BONDAGE RULE | 21 |
| TILE COORDINATE SYSTEM | 22 |
| TILE HIT TEST ALGORITHM | 22 |
| ATTRIBUTES FOR RENDERING ORDER | 24 |
| 6. SOME METHODS | 24 |
| DYNAMIC ARRAY | 24 |
| PARSER | 25 |
| 256 COLOR PALETTES IN UNICHAT 2 | 26 |
| PREDEFINED PALETTE INDICES | 27 |
| HOW TO MAKE A DIALOG WITH 256-COLOR IMAGE | 28 |
| MFC SDI ARCHITECTURE | 31 |
| CLASS HIERARCHIES | 34 |

1. Source Files

Compiler: MS Visual C++ 4.0 or above

Number of Lines = 5,256 (UC2Ani.dll) + 19,094 (UC2.exe) + 3,962 (MapEd.exe) = 28,312 lines

| VC++ Project | Description | Module |
|--------------|----------------------------------------------|-------------|
| UC2Ani | UniChat Animation Library | UC2Ani.dll |
| UC2 | UniChat 2 Client System | UC2.exe |
| MapEd | Map Editor | MapEd.exe |
| UDSGen | UniChat Data Source Generator | UDSGen.exe |
| MonTool | Monitoring Tool for counting current members | MonTool.exe |

Directory Structure in the UniChat 2 Source CD-ROM

| Directory Name | Description |
|----------------|-------------------------------------------------------------------------------|
| UC2 | Root |
| +AsyncMoniker | Asynchronous Moniker (for files download) sample program |
| +BMC | Bitmap file manipulation sample code for CDIB (with separate palette file) |
| -Debug | DEBUG: intermediate files for the compiler |
| -Doc | Some documents |
| +GenSDI | Generic SDI: Visual C++ compiler's generated template for an SDI project |
| -Hlp | Help project files for UniChat 2 |
| +ImageSrc | Image Sources for UniChat 2 |
| +I_Eng | Resource Definition files for English version |
| -I_Kor | Resource Definition files for Korean version |
| +I_Mall | Resource Definition files for Mall version |
| +MapEd | Map Editor Project |
| +Data | Data files: The default directory for data in UniChat 2 debugging environment |
| +MonTool | Monitoring Tool project |
| +Progress | Test project for data downloading with progress control view |
| -Release | RELEASE: intermediate files for the compiler |
| -Res | RES: Resource files for UniChat 2 |
| +Setup | Setup project for UniChat 2 |
| -System | Redistributable files for UniChat 2 |
| +Test | Test project |
| +UC2Ani | Animation library project |
| +UDSGen | UniChat Data Source file Generator Project |
| +WBTest | Web Browser Test Project |

Project UC2Ani which generates UC2Ani.dll

| Class | Base Class | Description | H | CPP |
|---------------------------|-------------------|----------------------------------------|-----|-------|
| CDBPal | :CPalette | Palette Manager | 35 | 386 |
| CDB | :CObject | Device Independent Bitmap manipulation | 174 | 1,548 |
| CSprite | :CDB | Bitmap Sprite | 133 | 329 |
| CPhasedSprite | :CSprite | Phased Sprite Animation | 97 | 397 |
| COSBView | :CScrollView | Off-Screen Buffered View | 87 | 421 |
| CSpriteList | :CObList | Sprite List structure | 38 | 173 |
| CSpriteNotifyObj | :CObject | Notification Object for Csprite | 38 | - |
| CSpriteListNotifyObj | :CSpriteNotifyObj | Notification Object for CSpriteList | 36 | 65 |
| CBubble | :CObject | Bubble Drawing | 64 | 287 |
| CBubbleList | :CObList | Bubble List structure | 38 | 149 |
| CBubbleNotifyObj | :CObject | Notification Object for CBubble | 36 | - |
| CBubbleListNotifyObj | :CBubbleNotifyObj | Notification Object for CBubbleList | 34 | 65 |
| CMCObject | :CObject | Media Control Interface | 40 | 213 |
| CSound | :CObject | Simple Sound Player using MCI | 37 | 131 |
| CPSButton | :Cbutton | Phased Sprite Button | 62 | 143 |
| Unichat Animation Library | | | 949 | 4,307 |

5,256

Project MapEd which generates MapEd.exe

| Class | Base Class | Description | H | CPP |
|-----------------|----------------|------------------------------------------|-----|-------|
| CAboutDlg | :CDialog | About Dialog | - | - |
| CActor | :CPhasedSprite | Actor | - | - |
| CActorDesc | | Actor Description | - | - |
| CBehavior | | Behavior of actors | - | - |
| CBaseDlg | :CDialog | Base Dialog | - | - |
| CFlatToolBar | :CToolBar | Toolbar for flat buttons | 32 | 245 |
| CGetIntDlg | :CDialog | InputDialog for Integer | 49 | 60 |
| CGetTextDlg | :CDialog | InputDialog for Text | 48 | 53 |
| CMainFrame | :CFrameWnd | MainFrame Window for MapEditor | 70 | 239 |
| CMapEdApp | :CWinApp | Application Window for MapEditor | 66 | 201 |
| CMapEdDoc | :CDocument | Document Class for MapEditor | 138 | 589 |
| CMapEdView | :COSBView | View Class for MapEditor | 125 | 1,054 |
| CMapEnvDlg | :CDialog | InputDialog for Map Environment variable | 60 | 78 |
| CMapListView | :CFormView | View for the list of tiles and sprites | 113 | 489 |
| CMemberInfo | | Member Info | 89 | 164 |
| CParser | | Line by line parser for the text file | - | - |
| CResMan | :CObject | Resource Manager | - | - |
| CStage | :CObject | Stage | - | - |
| CTextFileBuffer | :CObject | Memory management for text files | - | - |
| CTileMap | :CObject | Tile and map management | - | - |
| Map Editor | | | 790 | 3,172 |

3,962

Project UC2 which generates UC2.exe

| Class | Base Class | Description | H | CPP |
|---------------------|----------------------|---------------------------------------|-------|--------|
| CActor | : CPhasedSprite | Actor | 91 | 406 |
| CActorDesc | | Actor Description | 31 | 122 |
| CBaseChannel | : CObject | Base ChannelService | 101 | 467 |
| CBaseSocket | : CObject | Base SocketService | 112 | 621 |
| CBehavior | | Behavior of actors | 94 | 138 |
| CBindStatusCallback | : BindStatusCallback | Callback class for downbading | 63 | 108 |
| CBaseDlg | : CDialog | Cbase Dialog | 71 | 280 |
| CDownInfo | | Downbad info | 25 | 65 |
| CDownload | | Downbad class using Monker | 17 | 308 |
| CEditHistory | : CEdit | Edit control for History Panel | 51 | - |
| CEditSend | : CEdit | Edit control for Sending text | 36 | - |
| CInputPassword | : CDialog | Input Password Dialog | 47 | 44 |
| CLoginDlg | : CDialog | Login Dialog | 144 | 920 |
| CMainFrame | : CFrameWnd | MainFrame Window for UnChat | 127 | 860 |
| CMemberInfo | | Member Info | 89 | 164 |
| CMemberListDlg | : CDialog | Member List Dialog | 85 | 406 |
| CParser | | Line by line parser for the text file | 135 | 422 |
| CPPActor | : CPropertyPage | Property Page for actor | 76 | 317 |
| CPPChannel | : CPropertyPage | Property Page for ChannelList | 89 | 493 |
| CPPCreateChannel | : CPropertyPage | Property Page for Create Channel | 63 | 166 |
| CPPMemberInfo1 | : CPropertyPage | Property Page 1 for Member Info | 88 | 94 |
| CPPMemberInfo2 | : CPropertyPage | Property Page 2 for Member Info | - | - |
| CProgressDlg | : CDialog | Dialog for downbading | 82 | 629 |
| CPSFrame | : CMainFrameWnd | Frame Window for member property page | 51 | 103 |
| CPSJoinChannel | : CPropertySheet | Property Sheet for Joining Channel | 106 | 375 |
| CPSMemberInfo | : CPropertySheet | Property Sheet for Member Info | 52 | 52 |
| CResMan | : CObject | Resource Manager | 156 | 1,252 |
| CSplashWnd | : CWnd | Splash Window for 256 color image | 66 | 237 |
| CStage | : CObject | Stage | 141 | 1,099 |
| CTextFileBuffer | : CObject | Memory management for text files | 136 | 423 |
| CTileMap | : CObject | Tile and map management | 156 | 1,197 |
| CUC2App | : CWinApp | UnChat2 Application Window | 96 | 500 |
| CUC2Channel | : CBaseChannel | ChannelService for UnChat | 35 | 148 |
| CUC2Doc | : CDocument | Document class for UnChat | 186 | 1,575 |
| CUC2History | : CDialogBar | History Panel | 66 | 74 |
| CUC2Panel | : CDialogBar | Control Panel | 75 | 271 |
| CUC2Socket | : CBaseSocket | SocketService for UnChat | 65 | 384 |
| CUC2View | : COSBView | View class for UnChat | 127 | 893 |
| CWhisperDlg | : CDialog | Whisper Dialog | 48 | 46 |
| | | UnChat2 Messages Definition | 156 | |
| UnChat2 Client | | | 3,435 | 15,659 |

19,094

2. Data Files

| File Type | Description | Example |
|-----------|---------------------------------|-------------------|
| RIT | Resource Information Table | U2Res000.rit |
| SIT | Stage Information Table | 0000csin.sit, ... |
| UDS | UniChat Data Source (like a DB) | a00.uds, ... |





RIT and SIT files are distributed after LZ compression. UniChat client program can cleverly load these files whether they are compressed or not.

RIT – Resource Information Table

RIT contains all the definitions for the UniChat resources. UniChat resources are mainly tiles, static sprites, animated sprites, avatars and animation descriptions. For example, every static sprite needs an information for the bottom center position in the image besides the image file itself.

The client system does not store each file name for the sprite but it sequentially generates serial numbers from 0 and use this number as an identifier to the sprite image.

For images, we have the following types:

| Object Type | Sample File | Naming and Scripting |
|-------------------------------------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tile SPRITE_TILE SPRITE_WALL |  | tNAMEnnn.bmp t00 city001=(1,7); // t00 is a unichat data source file // cells are arranged in 1 column and 7 rows |
| Static Sprite SPRITE_STATIC |  | cNAMEnnn.bmp cs00 ctree009=(29,83); // center of the bottom (earth position) – in pixel units // the crossing point at the left image indicates this position (In Map Editor, this position is shown in this way) |
| Animated Sprite SPRITE_PHASED |  | iNAMEnnn.bmp cw00 ircha001=(3,1),(20,36); // cell dimension, center of earth position |
| Avatar SPRITE_ACTOR |  | aNAMEnnn.bmp a00 aman_001=(11,4),(23,45); // cell dimension, center of earth position |

- Every image sets its first pixel, the left top pixel of the image, as a transparent color.
- Static sprites and animated sprites can have some linked URLs in the script.

You can assign 3 types of animations for each sprite in Map Editor. CSprite::GetAniType() return the associated animation type of the sprite. In CPhasedSprite::HeartBeat(...), you can see the code for these animations.

| Animation Type | Description |
|--------------------------|----------------------------------------------------------|
| SPRITE ANI REPEAT | Cell animation with assigned interval |
| SPRITE ANI FADE | Animation effect with fade-in and fade-out of one cell |
| SPRITE ANI RANDOM | REPEAT animation but with random interval between cycles |
| SPRITE ANI ACTOR | Actor keeps animation data in CActorDesc and CBehavior |

```
; "u2res00.rit"
;=====
; Resource Info Specification
; RIT: Resource Information Table

;=====
; filename=[(col,row)][,(cz.x,cz.y)][,nickname];
; filename = resource type flag(1) + name(2) + serial(3);
; resource type flag = t(TILE), s(STATIC), i(ANI), a(AVATAR), w(WAVE), m(MIDI)
; (col,row) = # of cells in (column, row); for STATIC this attribute is omitted
; (cz.x,cz.y) = center of z-order in (x, y) pixel position from the leftop
; nickname = nickname for actor
#VERSION=0.99;
// Think of tile as a PhasedSprite
#TIL=      // TILE
{
t00|tcity001=(1,7);      // "tcity001.bmp" in "t00.uds" has 7 cells in one row.
t00|tcity002=(1,5);
t00|tcosm001=(3,7);
...
}
#STT=      // STATIC Sprites - These files comprise of only one cell.
{
    // bottom center position (cz.x, cz.y) in pixel
ca00|cadve001=(10,78),http://www.unitel.co.kr/;
ca00|cadve006=(23,115),http://www.mcdonald.com/;
ca00|cbill001=(32,98);
...
}
#ANI=      // ANIMATED Sprites
{
cw00|iadve001=(3,1),(110,0),http://a.b.c/unichat/http://a.b.c/http://www.naver.com/;
cw00|iadve002=(4,1),(1,6),http://a.b.c/http://www.samsung.co.kr/http://a.b.c/http://www.naver.com/;
cw00|iadve003=(4,1),(1,6),http://a.b.c/http://a.b.c/http://a.b.c/unichat/http://www.naver.com/;
cw00|iarro001=(5,1),(5,0);
...
}
#AVT=      // AVATAR
{
a00|aman_001=(11,4),(23,45);
a00|aman_002=(11,4),(23,45);
a00|aman_003=(11,4),(23,45);
```

```

...
}
#WAV=    // Wave files
{
sagry000; // sagry000.wav
schng000;
...
}
#MID=    // MIDI files
{
mjazz000; // mjazz000.mid
mjazz001;
...
}
#STAGE=  // User-creatable stages
{
0000csin; // 0000csin.sit
0001ctrm;
...
}
#SERVERIP=    // Server IP Addresses
{
203.241.132.83;    // LAN
88.1.26.2;         // MODEM
127.0.0.1;
ComicSrv1.Microsoft.Com;
vchat1.microsoft.com;
}

; ACTOR SECTION
; (*i): - change orientation, * 50%, / 25%, | 12%, ~ 0%, ^ don't USE_COLORKEY
; #ACTOR=id,nickname,nMSPT;    // nMSPT: milliseconds per tick
; {
;     // nMSPT is Milliseconds per Tick
;     // Behaviors
;     0=nRepeatCount,(cell index(, delay ticks(, displacement_x, displacement_y(, sound id))))...;
;     // nRepeatCount = 0 for infinite loop,- value for pendulum movement
;     // delay ticks; 5 = fast; 10 = moderate; 15 = slow
; }

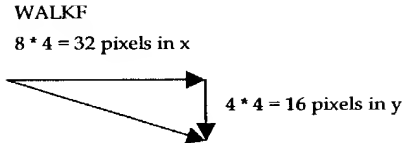
#ACTOR=a00|aman_001,man001,40;    // This first actor definition will be used for the default.
{
// Behaviors
STANDF=1,(0,2);    // Stand forward
STANDB=1,(9,2);    // Stand backward
STANDINGF=1,(|0,2,0,0,schng002)/(0,2)(*0,2)(#0,2)(0,2); // Appearing in forward stance
STANDINGB=1,(|9,2,0,0,schng002)/(9,2)(*9,2)(#9,2)(9,2); // sound id is a predefined wave file in RIT
MORPHF=1,(39,2);    // Morphed image

```

```

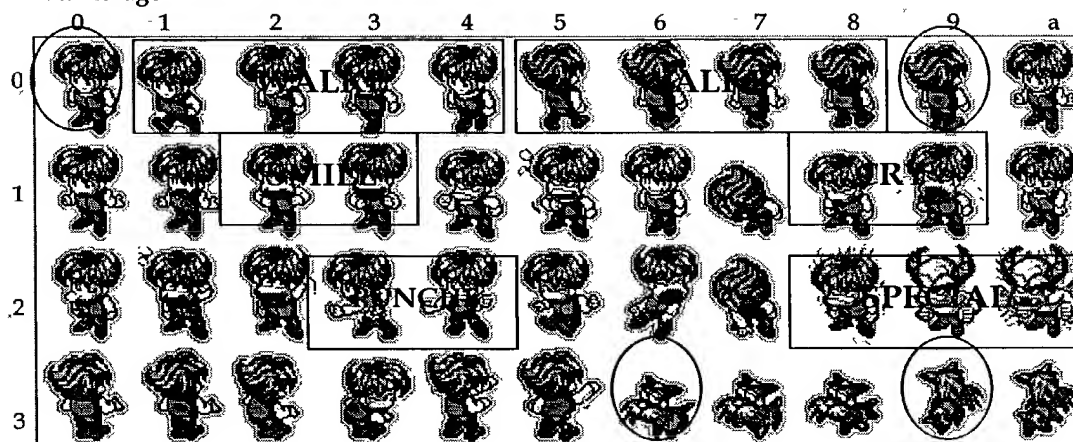
MORPHB=1,(42,2);
MORPHINGF=1,({39,2,0,0,schng000})/(39,2)(*39,2)(#39,2)(39,2); // Morphing sequence
MORPHINGB=1,({42,2,0,0,schng001})/(42,2)(*42,2)(#42,2)(42,2);
DOZEF=0,(*21,10)(*22,10);
DOZEB=0,(*33,10)(*34,10);
// Movements
WALKF=1,(1,5,8,4,sstep000)(2,5,8,4)(3,5,8,4)(4,5,8,4);
WALKB=1,(5,5,8,4,sstep000)(6,5,8,4)(7,5,8,4)(8,5,8,4);
UPF=1,(1,3)(2,3)(3,3)(4,3,0,0,sjpup000); // Jump up
UPB=1,(5,3)(6,3)(7,3)(8,3,0,0,sjpdn001);
DOWNF=1,(1,3)(2,3)(3,3)(4,3,0,0,sjpdn000);
DOWNB=1,(5,3)(6,3)(7,3)(8,3,0,0,sjpup001);
MORPHWALKF=1,(40,5,8,4,sturn000)(41,5,8,4)(40,5,8,4)(41,5,8,4);
MORPHWALKB=1,(42,5,8,4,sturn001)(43,5,8,4)(42,5,8,4)(43,5,8,4);
// Gesture Commands
CHAT=3,(10)(11)(12);
ENTER=1,({0,3,0,0,sstep000})/(0,3)(*0,3)(#0,3)(0,1);
EXIT=1,({0,3,0,0,sstep000})/(#0,3)(*0,3)/(0,3)(0,3);
SMILE=1,(13,0,0,stemp000)(14)(13)(14)(13)(14);
MAD=1,(15,5,0,0,sagry000)(16)(15)(16)(15)(16);
HELLO=1,(17,10)(18);
CRY=1,(19,5,0,0,scrys000)(20)(19)(20)(19)(20);
SCRATCH=1,(23,3,0,0,stemp001)(24,2)(23,3)(24,2);
PICK=1,(29,10,0,0,spick000);
SPECIAL=1,(30,5,0,0,stemp000)(31)(32)(*32)(32)(31)(32);
WIGGLEB=2,(33)(34);
PUNCHF=3,(25,5,0,0,shit_000)(26);
PUNCHB=3,(37,5,0,0,shit_002)(38);
BEATENF=1,(27)(28,2)(*28,3)(28,2)(27,3);
BEATENB=1,(35)(36,2)(*36,3)(36,2)(35,3);
;TURN180
;TURN360
}
#ACTOR=a00|aman_002,man002,20; // Overload some definitions
{
MORPHWALKF=1,(40,5,8,4,sstep000)(41,5,8,4)(40,5,8,4)(41,5,8,4);
MORPHWALKB=1,(42,5,8,4,sstep000)(43,5,8,4)(42,5,8,4)(43,5,8,4);
SPECIAL=1,(30,5,0,0,stemp000)(31)(32)(31)(32);
}
#ACTOR=a00|aman_003,man003,50; // will use actor definition of the first one a00|aman_001 as a default
{
}
...

```



In the ACTOR section, actor's behaviors are clearly described by some kind of script. For each predefined behavior, we associate a sequence of cell animation and some parameters.

Avatar Image



In UniChat 2, every actor image has the same number ($11 \times 4 = 44$) of cells in it and each cell is associated with the predefined behavior.

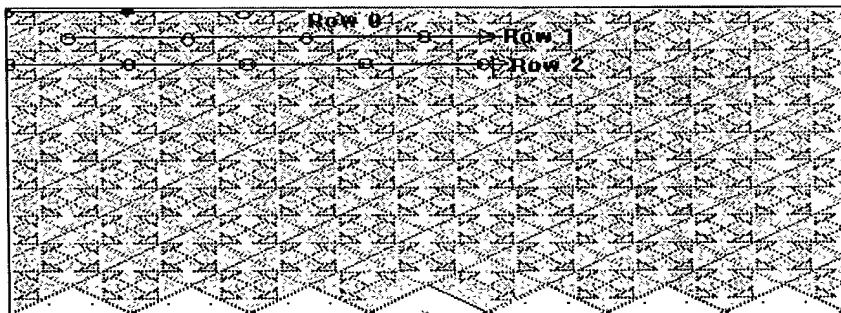
SIT – Stage Information Table

SIT is a file format for storing each background's map and objects on it. It has two sections, one for a list of sprite objects and other for the tile map. One background needs one SIT file. These files are automatically generated by Map Editor tool.

```
; Stage 0030mall.sit
#VERSION=1.01;
// Sprites data
// #STAGE=[room id] Title, Background Music Sequence;
#STAGE=[u2/0030mall]The M@ll-The Biggest Shopping Center on the Earth,5>9>1>4>2>8>;
{
    // Cstage::Load()
    // Resource ID = Cell ID, LeftTop position, Image Operation, Elevation, Sprite Type, nMSPT;
    333=0,(134,-65),0,0,65535,0;
    257=0,(454,-69);
    335=0,(314,-43),256,0,65535,0;
    257=0,(497,-49);
    224=0,(616,4),256,0,4,0;
    242=0,(106,-57);
    257=0,(540,-27);
    389=0,(317,-65),256,0,65535,0;
    ...
} // 100 sprites.
// MAP data
#TILESIZE=(64,32); // Tile Size in pixel
#SCREENSIZE=(640,368); // Dimension
#ROW=(0); // CtileMap::LoadRow()
{
```

```
// -1 = Actor Elevation, Direction Attribute;      // for NULL tile (that has no associated tile image)
// Resource ID = Cell ID, Elevation, Image Operation, Actor Elevation, Sprite Type, nMSPT, Direction Attribute, Hyperlink;
1=3;
1=0;
3=1;
3=1,0,0,0,1,0,13;
1=3,0,0,0,1,0,12;
1=3,0,0,0,1,0,0;
1=3,0,0,0,1,0,0;
3=2,0,0,0,1,0,14;
3=2;
1=0,0,256,0,1,0,15;
0=2;
}
#ROW=(1);
{
...
-1=0,9;
}
...
#ROW=(23);
{
0=1,0,0,0,1,0,15,x:0020mall;      // This tile is an exit with link to "0020mall.sit"
...
}
; 264 tiles.
```

Tiles are managed by rows in the memory and graphically their sequence in the screen looks like this picture.

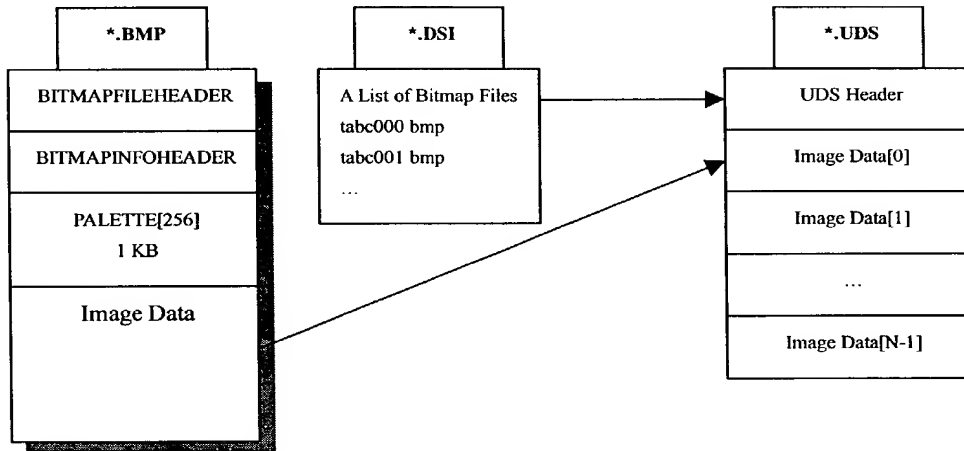


UDS - UniChat Data Source

UniChat 2 has more than 360 bitmap image files. In order to optimize disk usage by reducing the number of files, we combine multiple files into one data source file. This data source file has its own index table in it and actual bitmap images. And to reduce the file size UDS file does not store each palette table of the image. Since we are using one master palette with 256 colors, we

don't have to have the palette for each image redundantly.

UDSGen.exe is a tool to merge bitmap images into one file.



CDIB class in UC2Ani library knows how to read UDS file and load a specified bitmap image in it.

Naming rule for the resource location:

Data Source(*.uds) | Bitmap Filename(*.bmp)

For example,

CDIB* pDIB;

...

pDIB->Load("a00 |atree001");

// In "a00.uds" file, find "atree001.bmp" image and load into memory

3. Data Model

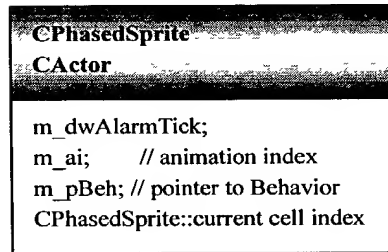
The model of Classes in UniChat is an analogy of the stage and the actors on it. We can think of animation as a play of the actors on the stage.

Besides the basic classes in UC2Ani.dll, the animation related classes are the following:

| Class | Description |
|------------|-----------------------------------------------------------------------|
| CResMan | Resource Manager |
| CStage | Stage has control for creation of tiles, actors and sprites. |
| CFileMap | Loads tile data from .SIT and makes an image for the background |
| CBehavior | Behavior is a sequence of data for cell animations |
| CActorDesc | Actor Description class keeps a record for each predefined characters |
| CActor | Actor represents each user in UniChat |

Animation Mechanism

Here is the animation mechanism. CStage::TickAll() function is a heart that pumps all the animations in the system. Behavior index and alarm tick in CPhasedSprite and CActor are some kind of switches. CActor is inherited from CPhasedSprite that has basic animation functions. Since CActor is for actors with various behaviors like walking, smiling, and crying, CActor involves more data and methods to deal with.



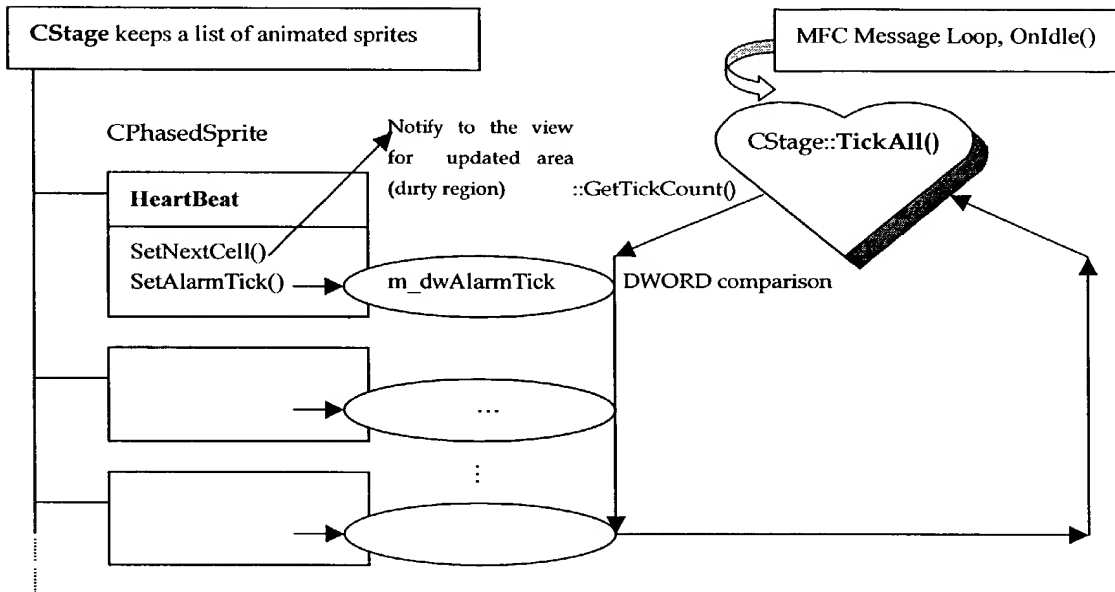
```

void CPhasedSprite::HeartBeat(DWORD dwCurTick)
{
    ...
    SetNextCell(); // Advance current cell index
    SetAlarmTick(dwCurTick + GetMSPT()); // Set alarm for the next animation
    ...
}

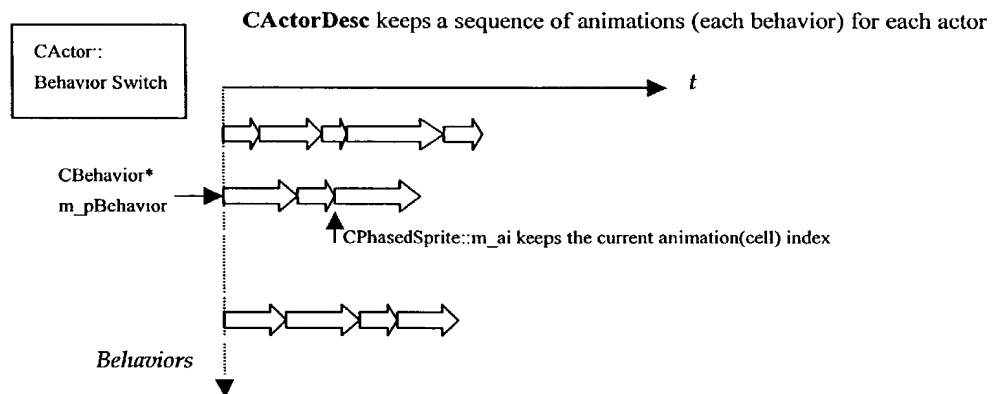
void CPhasedSprite::SetNextCell()
{
    // Notify the change of its image to the current view (COSBView)
    m_pNotifyObj->Change(this, CSpriteNotifyObj::IMAGE, &rcPos);
}

int CStage::TickAll()
{
    DWORD dwCurTick = ::GetTickCount();
    POSITION pos = m_AniList.GetTailPosition();
    while (pos)
    {
        CPhasedSprite* pPS = (CPhasedSprite*)m_AniList.GetPrev(pos); // Increment position.
        if (dwCurTick >= pPS->GetAlarmTick())
        {
            if (pPS->HeartBeat(dwCurTick))
                m_pOSBView->RenderAndDrawDirtyList();
        }
    }
    ...
}
  
```


In MFC's OnIdle(long) function, this CStage::TickAll() function is called automatically.

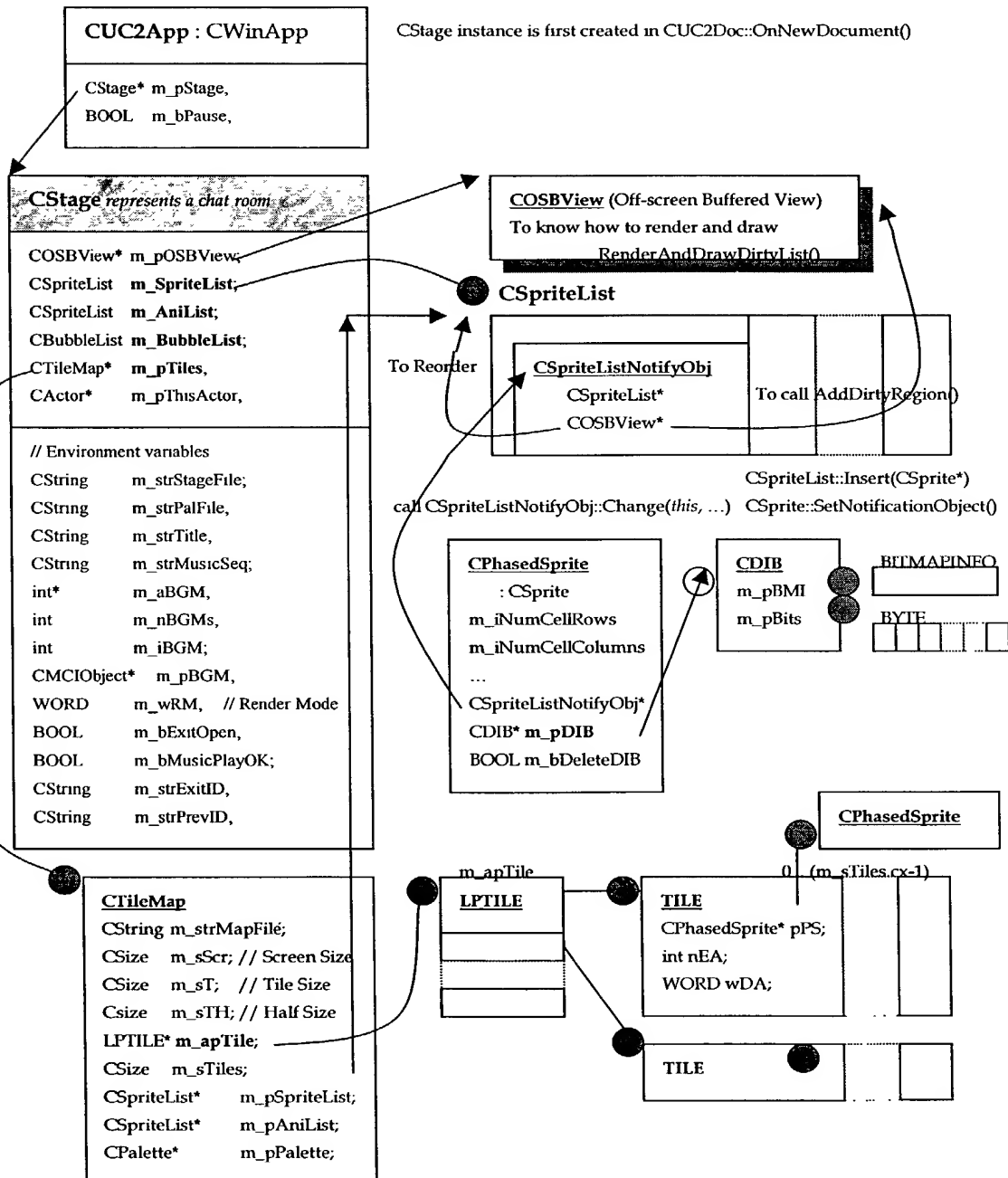
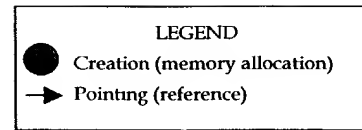


For actors, since each actor has several behaviors involved, we need a kind a switch to designate current behavior.



Memory model for the Stage and Actors

BOOL CUC2Doc::OnNewDocument()



Each sprite notifies its change of state to the view to request redrawing like the following

example code.

```

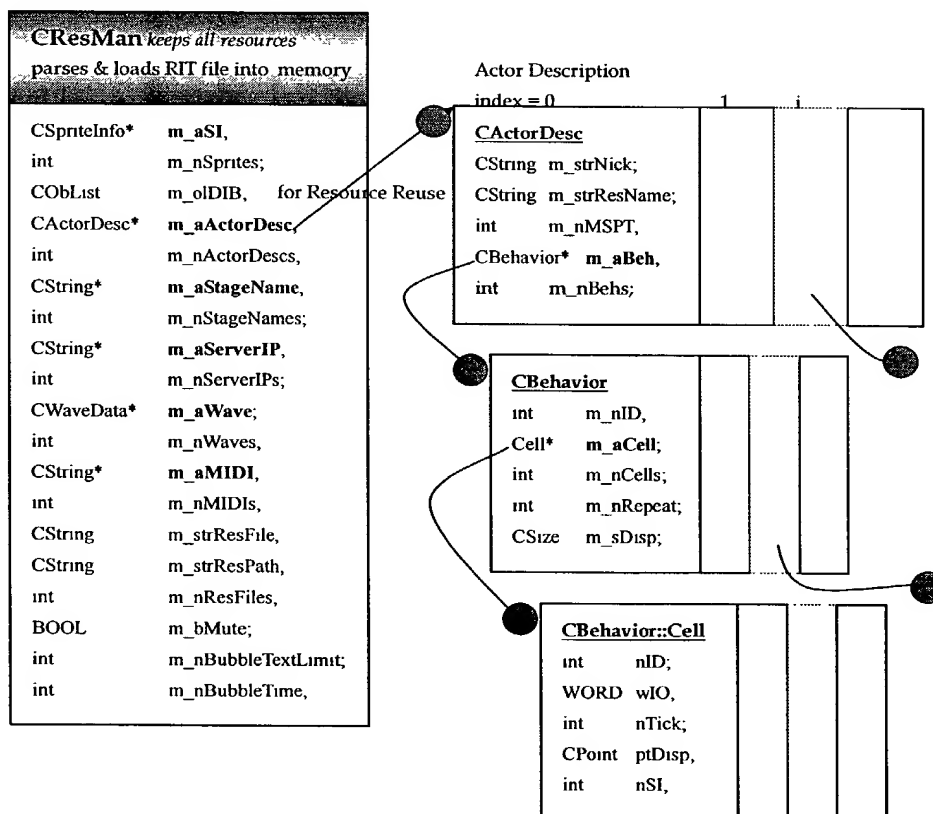
CSprite::SetZ()
    m_pNotifyObj->Change(this, CSpriteNotifyObj::ZORDER, &rc);

void SpriteListNotifyObj.Change(CSprite*, CHANGETYPE, CRECT*, CRECT*)
{
    if (change & CSpriteNotifyObj::ZORDER)
    {
        m_pSpriteList->Reorder(pSprite);
        m_pBufferView->AddDirtyRegion(pRect1);
    }
    ...
}

```

Resource Manager and Actor Description Tables

For actors, CResMan loads actor animation data from RIT file into an array of CActorDesc objects.



Here is an example of Actor Description:

Actor Description

```
#ACTOR=a00|aman_001,man001,40;           // This first actor definition will be used for the default.
{
    // Behaviors
    STANDF=1,(0,2);           // Stand forward
    Behavior → STANDB=1,(9,2); // Stand backward
    STANDINGF=1,([0,2,0,0,schng002]/(0,2)(*0,2)(#0,2)(0,2); // Appearing in forward stance
    STANDINGB=1,([9,2,0,0,schng002]/(9,2)(*9,2)(#9,2)(9,2); // sound id is a predefined wave file in RIT
    ...
}
```

There is a bitmap resource reuse mechanism in CResMan. By using CResMan::LoadDIB() and CResMan::LoadPhasedSprite() one can prevent loading the same bitmap image into memory redundantly. CResMan keeps a list of resource names that are already loaded into memory. So, for any request to load an image with the same resource name, CResMan just returns to the pointer to the memory without actually allocating memory.

4. Communication Architecture using MS ChatSock

MS ChatSock API

Microsoft provides ChatSock API to support for MIC (Microsoft Internet Chat) protocol. This protocol and APIs are well documented in their SDK. UniChat's communication architecture totally depends on this ChatSock API. For example, our actor in UniChat is implemented in CActor class and there is an interface called ICSMember for each chat client. Then CActor keeps a member for ICSMember to use ChatSock functionalities. So, it is important to identify which object is for ChatSock interfaces and which is for UniChat classes.

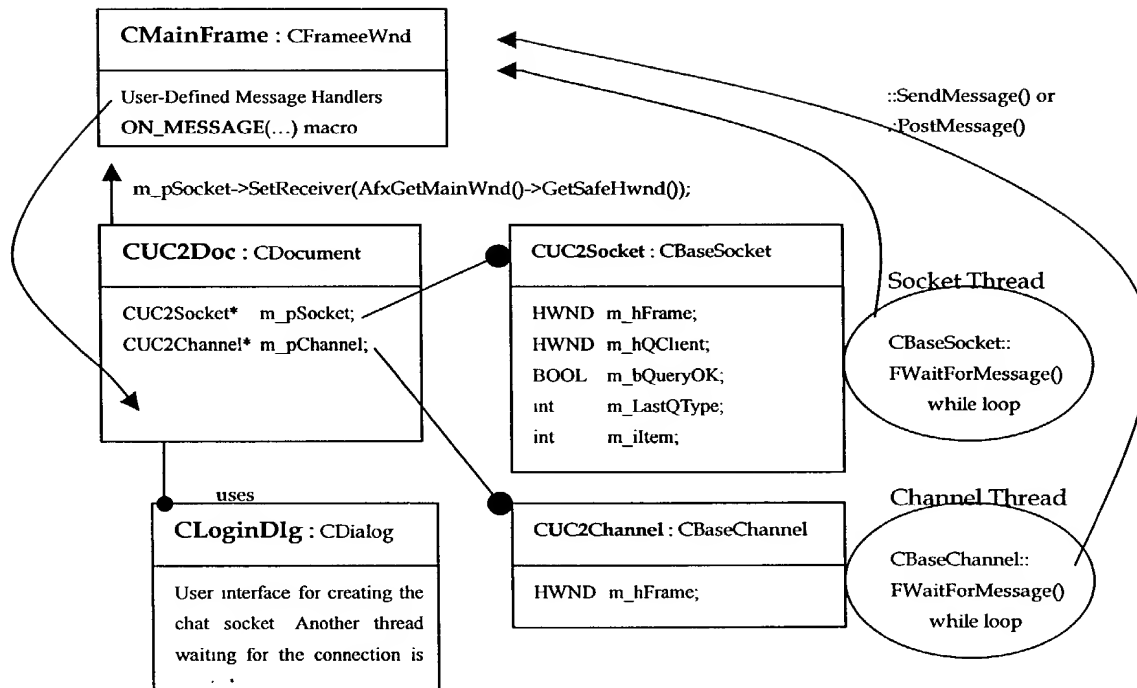
Since ChatSock is implemented on the COM (Component Object Model) architecture, programmers have to understand some basic concepts for COM programming, like interfaces, AddRef(), Release().

One thing that confuses the programmers in ChatSock API is the sequence of messages and events. This is not documented so you have to run some sample and trace some events to find out the flow of messages. Here is a diagram that shows how UniChat uses ChatSock messages when a new member joins a channel.

ChatSock-related classes

| Class | Description |
|--------------|-------------------------------------------------------|
| CBaseSocket | Base Socket functionalities provided by ChatSock |
| CBaseChannel | Base Channel interface provided by ChatSock |
| CUIC2Socket | Inherited from CBaseSocket, UniChat 2 specific codes |
| CUIC2Channel | Inherited from CBaseChannel, UniChat 2 specific codes |

Integration with MFC classes



I made CUC2Doc create and keep the pointer to ChatSock services. When you have a need to use ChatSock services, you have to get the pointer to ChatSock from this document object.

ChatSock knows where to send its messages by *m_hFrame*. This handle is set by CUC2Socket::SetReceiver(const HWND hWnd) in CUC2Doc class. If this is set to NULL, ChatSock doesn't send its message. Because ChatSock is running in another thread, you have to be sure to call SetReceiver(NULL) when you're not ready to process the messages.

Socket is created in BOOL CUC2Doc::Connect() through CLoginDlg dialog and channel is created in BOOL CUC2Doc::SetChannel(PICS_CHANNEL picsChannel).

One problem using ChatSock in MFC is that socket services are running in other threads. And in MFC windows, GDI objects, and other objects should be passed between threads by means of handles instead of by means of pointers to MFC objects. That's why I put the handle *m_hFrame* in CUC2Socket and CUC2Channel. ChatSock sends its messages through this handle. But an object of CUC2Doc class that is inherited from CDocument is not a normal window - CDocument has no HWNDs.

So we have to use CMainFrame object as a receiver for socket services. You can see many user-defined message handlers in this class just calling relevant handlers in CUC2Doc. Here is a

segment of that code.

```
// MainFrm.h
afx_msg LRESULT OnCsData(WPARAM, LPARAM);
...
// MainFrm.cpp
ON_MESSAGE(CSMMSG_CMD_DATA, OnCsData)
...
LRESULT CMainFrame::OnCsData(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsData(wParam, lParam);
}
```

It is recommended to read the section on “Sharing MFC Objects Among Threads” in Chap. 14 of Prosiš’s book. (Programming Windows 95 with MFC, Microsoft Press)

Here is a list of message handlers triggered by the ChatSock service.

```
// MainFrm.h
afx_msg LRESULT OnCsAddChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsPrivateMsg(WPARAM, LPARAM);
afx_msg LRESULT OnCsQueryData(WPARAM, LPARAM);
afx_msg LRESULT OnCsInvite(WPARAM, LPARAM);
afx_msg LRESULT OnCsGotMemList(WPARAM, LPARAM);
afx_msg LRESULT OnCsAddMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsDelMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsDelChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsModeMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsModeChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsTextA(WPARAM, LPARAM);
afx_msg LRESULT OnCsData(WPARAM, LPARAM);
afx_msg LRESULT OnCsWhisperText(WPARAM, LPARAM);
afx_msg LRESULT OnCsWhisperData(WPARAM, LPARAM);
afx_msg LRESULT OnCsNewTopic(WPARAM, LPARAM);
afx_msg LRESULT OnCsNewNick(WPARAM, LPARAM);
afx_msg LRESULT OnChannelFullRetry(WPARAM, LPARAM);
```

User-defined messages for the communication between ChatSock classes and our application windows are included in “UC2Messages.h” file.

Action Commands

Action commands for character movements or gestures are also enumerated in “UC2Messages.h” file.

I made a simple protocol to send and receive commands between clients. Each client packs some data into a NULL-terminated string as in the following syntax.

X`n` (additional data each separated by `)

For example,

"X`5`(3,19)`256" ... X: flag, 5: command id, (3,19): tile id, 256: state value

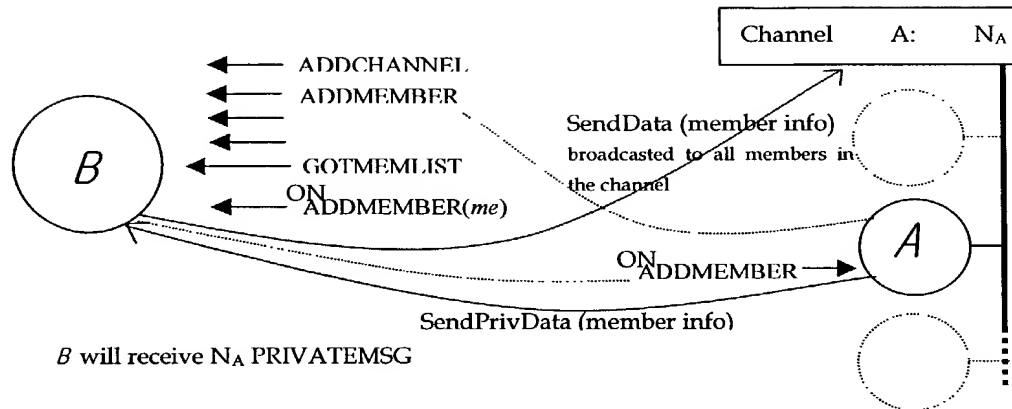
This is the command CMD_MOVEF=5. So the client that receives this data will parse the string and move the corresponding character. Since the receiving client already knows where the message came from, we don't need to add an identifier for the sender. The tile id is included to verify the position after movement. So each time for action commands character positions have a chance to synchronize their positions with other users.

```
enum ACTOR_COMMANDS
{ // Command Enumerators
  CMD_BEGIN = 0,
  // Management
  CMD_MEMBER_INFO, //
  X`nCmd`nVersion`nCharID`nBubbleKind`strHandle`strRealName`strProfile`ptTID`wState
  CMD_MEMBER_ACTOR, // X`nCmd`nCharID`nBubbleKind (changed his Actor)
  CMD_NEWS, // X`nCmd` message
  // Position Move
  CMD_MOVE,
  CMD_MOVEF, // X`nCmd`ptTID`wState
  CMD_MOVEB,
  CMD_RES_MOVE,
  // State Change
  CMD_STATE,
  CMD_STAND, // State
  CMD_MORPH,
  CMD_DOZE,
  CMD_TURNL,
  CMD_TURNR,
  CMD_RES_STATE, // reserved
  // Actions // CMD_ACTION is for Just repositioning message
  CMD_ACTION, // Y`nCmd`ptTID`wState` (to verify synchronization)
  CMD_CHAT,
  CMD_ENTER,
  CMD_EXIT,
  CMD_SMILE,
  CMD_MAD,
  CMD_HELLO,
  CMD_CRY,
  CMD_SCRATCH,
  CMD_PICK,
  CMD_SPECIAL,
  CMD_PUNCH, // Y`nCmd`ptTID`wState`NickTo`
```

```

    CMD_BEATEN,
    CMD_RES_ACTION,
    CMD_END
};

```



CUC2Doc calls CBaseChannel::FSendData() when the client needs to broadcast its message to all the members in the channel. This is used for sending command data which should be shared by all members in the same channel.

CBaseSocket::FSendPrivData() is for sending data only to a designated user. I used this method to send each client's data to a new member in the channel. Then the new comer will have these messages from other guys already in the channel so that he can gather informations of others.

5. Tile Theories

UniChat uses a new kind of data structure for the background drawing. In QuarterView graphics, background image is dynamically composed by a set of tiles. Use of tiles can dramatically reduce in file size for expressing various backgrounds. Owing to this technology, UniChat 2 was able to pack all of the files into 1.2 MB having more than 50 backgrounds.

Tile Attributes

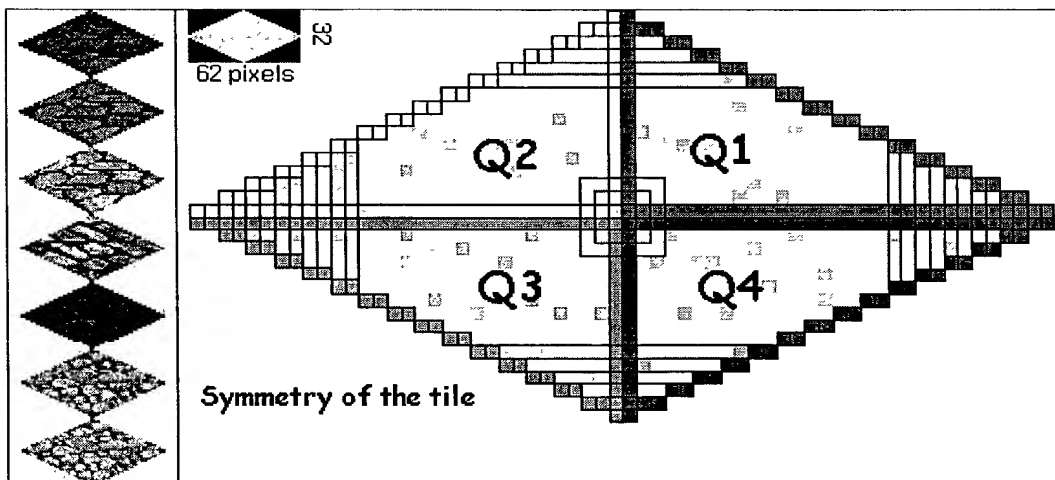
TILE

```

CPhasedSprite* pPS;
int    nEA;
WORD  wDA;

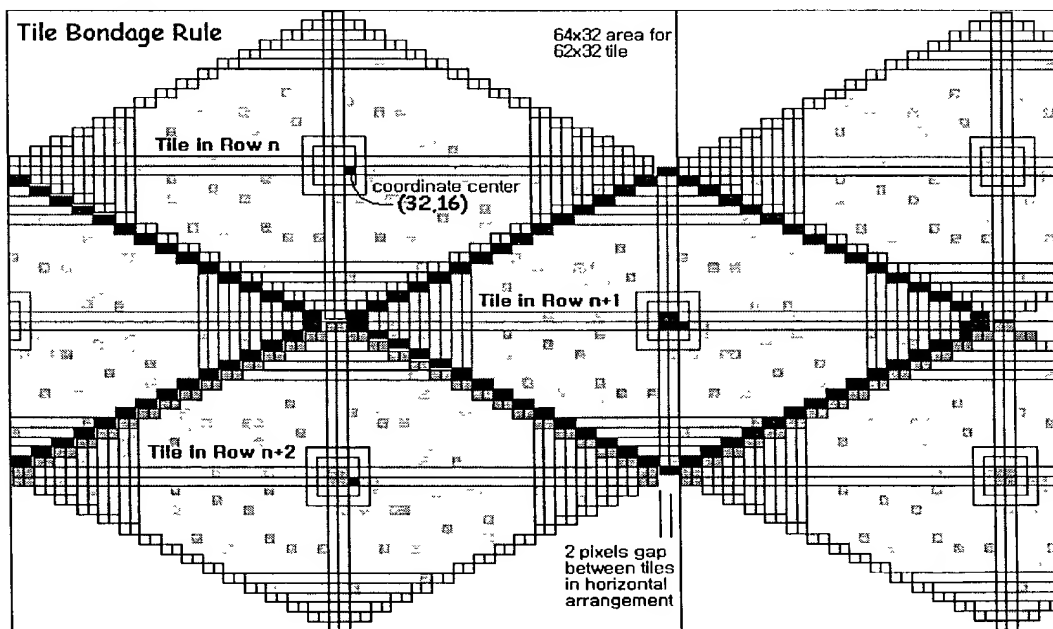
```

In our source code, Tile is defined as a PhasedSprite with two attributes for elevation and direction. CTileMap is a two dimensional array of these tiles. But the sprite for a tile should have the same size and shape as in the following picture:



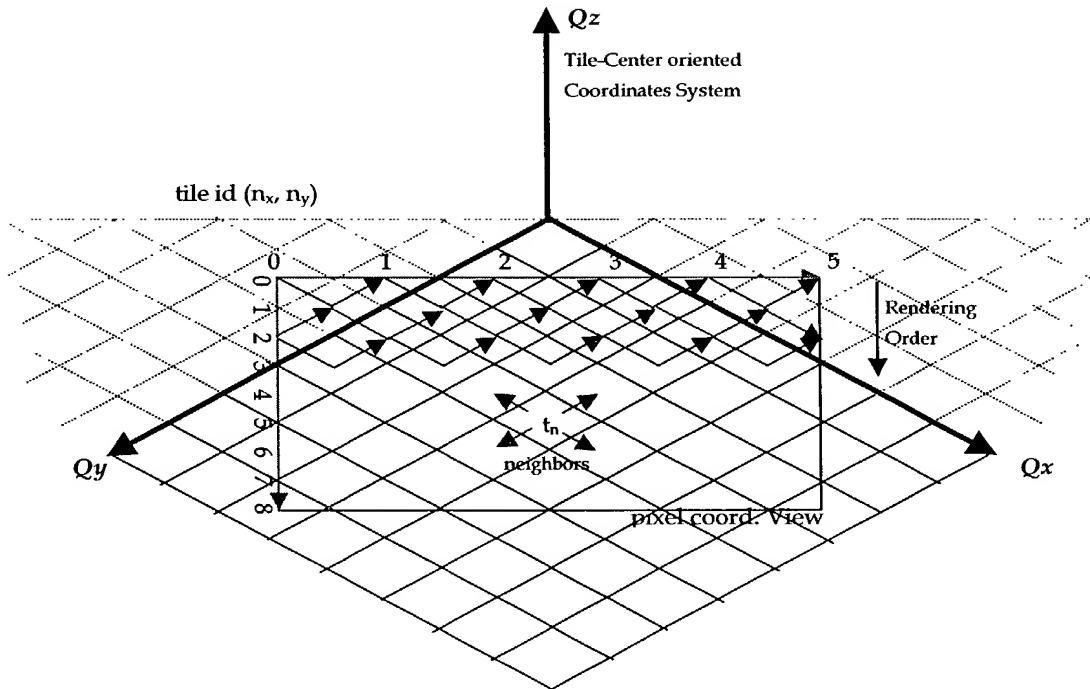
The left image shows many tiles packed into a file. By the horizontal and vertical symmetries of the tile structure, we can flip the image horizontally and vertically forming different images. But it doesn't have a rotational symmetry.

Tile Bondage Rule



Pixels actually engaged in the bondage are shown with different colors grouped by the rows.

Tile Coordinate System



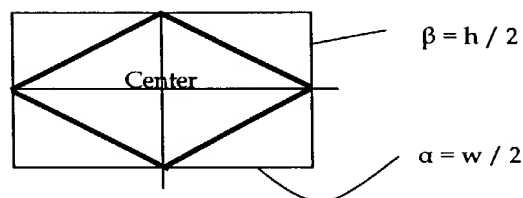
To find the neighboring 4 tiles of tile (n_x, n_y)

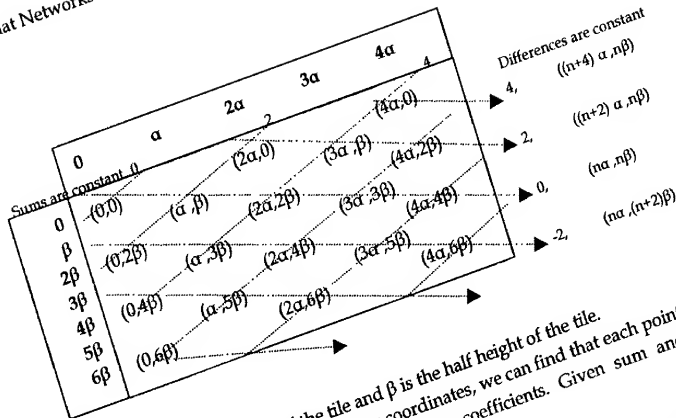
$$\begin{aligned}
 N(n_x, n_y) &= \{ (n_x-1, n_y\pm 1), (n_x, n_y\pm 1) \} \text{ for } n_y = 2n \text{ \& } \\
 &\quad \{ (n_x, n_y\pm 1), (n_x+1, n_y\pm 1) \} \text{ for } n_y = 2n+1 \\
 &= \{ (n_x-(-1)^{n_y}, n_y\pm 1), (n_x, n_y\pm 1) \}
 \end{aligned}$$

Tile Hit Test Algorithm

Problem: Find the nearest tile index for a given (x', y') position in pixel coordinates.

According to our tile index system, the center positions of tiles in pixel coordinates are like the following:





Here, α is the half width of the tile and β is the half height of the tile. In this table of sums and differences of its coefficients. Given sum and difference of the coefficients of x and y , we can find a unique point.

The problem is to find the closest integer M and N that satisfies the following:
 Since the coefficient of x coordinate is x / α and that of y coordinate is y / β ,
 Sum of coef. $= x_c / \alpha + y_c / \beta = 2M$, where $M=0,1,2,\dots$ - (1)
 Difference of coef. $= x_c / \alpha - y_c / \beta = 2N$, where $N = \dots, -2, -1, 0, 1, 2, \dots$ - (2)

Above equations are only applied to the exact center position of each tile. So, for a given point (x', y') , that may not be a center position, we must solve the following inequalities:
 $2M-1 < x' / \alpha + y' / \beta \leq 2M+1$
 $2N-1 < x' / \alpha - y' / \beta \leq 2N+1$

Here we define $S = x' / \alpha + y' / \beta$, $D = x' / \alpha - y' / \beta$, and these cannot be integers.
 $(S-1)/2 \leq M < (S+1)/2$ - (3)
 $(D-1)/2 \leq N < (D+1)/2$ - (4)

These can be solved for M and N with the condition that they should be integers. So the problem reduces to finding such an integer that satisfies above inequalities. By adding and subtracting equations (1) and (2), we get
 $C = (\alpha(M+N), \beta(M-N))$ - (5)

This algorithm has been implemented as a function `C TileMap::GetNearestTileCenter(const CPoint& pt)`.

```
CPoint C TileMap::GetNearestTileCenter(const CPoint& pt) const
{
    float fx = float(pt.x)/m_sTH.cx; // x/a
    float fy = float(pt.y)/m_sTH.cy; // y/b
```

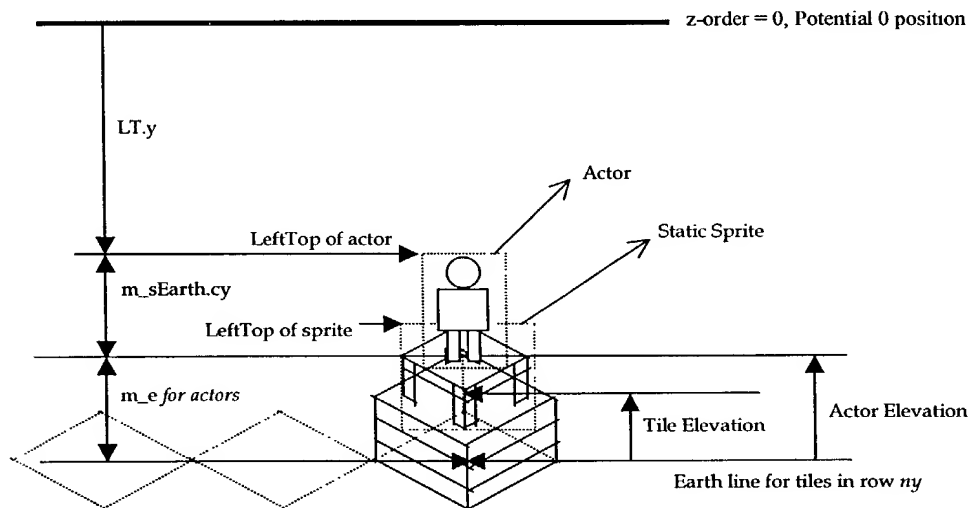
```

double fM = (fx + fy + 1.)/2.; // (S-1)/2 <= M < (S+1)/2
double fN = (fx - fy + 1.)/2.; // (D-1)/2 <= N < (D+1)/2 can be negative
if (fN <= 0.) // Consider a problem to find an integer that satisfies
    fN--; // -1.5 <= N < -0.5, But int(-0.5)=0
int M = int(fM);
int N = int(fN);
return CPoint(m_sTH.cx*(M + N), m_sTH.cy*(M - N));
}

```

Attributes for Rendering Order

The program has to know which sprite in the sprite list should be rendered first. This is the rendering order problem. In normal animation technique, every sprite has an attribute called *z-order* to indicate this order. But in quarterview this is not sufficient. Since we are actually in 3D coordinate space in quarterview, we need one more coordinate besides *x* and *y*. So I have introduced an attribute named *elevation*.



```

z = -GetEarthPointY();
= - (LT.y + m_sEarth.cy + m_e)

```

Z orders are the same for any actors on the tiles in the same row. Actually Z-order is determined by the tile on which the actor stands. This center of the tile is the *earth point*.

6. Some Methods

There are some commonly used methods in UniChat 2 source codes. It is some kind of personal programming style. But to understand the codes easily you have to identify these styles.

Dynamic Array

This is a method for allocating memories for a list of objects. If we can't determine the number of the objects at compile time, we cannot use static array definition in our code. Normally the number of objects is read from data file and determined by some calculations at runtime, so this method is very useful.

This method is just another usage of C pointer and some kind of naming rule. For example, to construct a list of string objects like the following in memory:

```
Data File    // SIT files
{
0000csin;    // 0000csin.sit
0001ctm;
...
0009ctm;
}

// Sample Source
class CSample
{
...
    CString* m_aSIT;
    int      m_nSITs;
}

CSample::Load()
{
    m_nSITs = parser.CountItems();    // Let's say parser.CountItems() returns the number of items in the data file
    m_aSIT = new CString[m_nSITs];
    for (int i=0, i < m_nSITs, i++)
        m_aSIT[i] = parser.ReadLine();
}

CSample. ~CSample()
{
    if (m_aSIT)
        delete [ ] m_aSIT;
}

```

This is the pattern used to handle this kind of dynamic memory.

| |
|------------------------------------------------------------------------------------------------------|
| <pre>TYPE* m_aObj; // pointer to an array of Obj's int m_nObjs; // the number of objects</pre> |
|------------------------------------------------------------------------------------------------------|

Parser

A text line parser is globally used in UniChat code to interpret our data files and load into the memory structured. CParser, once implemented in DOS environment, is a general class for this purpose modified for MFC classes. CTextFileBuffer is a utility class that enables CParser to use memory loaded files and LZ compressed files.

If you use CParser, it's very easy to count some items before dynamically allocating memory and to add some comment rules. Moreover some basic data types are interpreted according to the variable types with the same name of member function in CParser.

Here is a fragment of codes that show a typical use of this class:

```
extern CParser gParser;

BOOL CTestDoc::OnNewDocument()
{
    ...
    TRY
    {
        CStdioFile f("sample.txt", CFile::modeRead | CFile::typeText);
        char* szVal = new char[gParser.GetMaxBuffer()];
        int iVal;
        double fVal;
        CPoint ptVal;
        while(f.ReadString(szBuf, gParser.GetMaxBuffer()))
        {
            gParser.CopyBuffer(szBuf); // "Toto=1,3.4,(12,345);"
            if (gParser.IsCommentLine())
                continue;

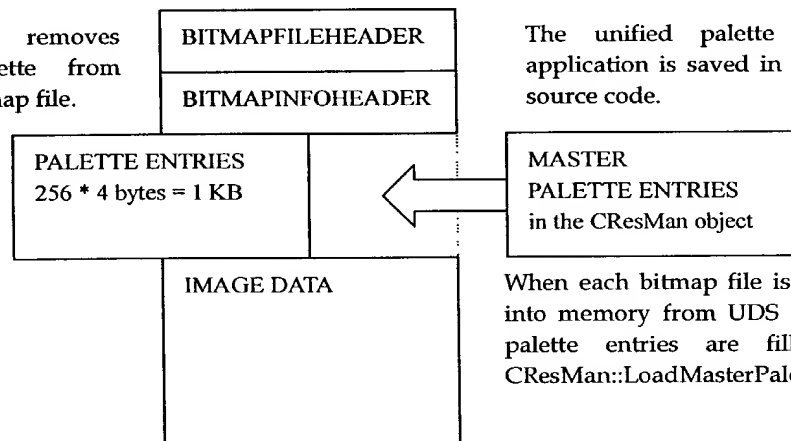
            gParser.GetValueRightToken(szVal, '=');
            gParser.SetLeftToken('=');
            gParser.GetValueRightToken(iVal, ',');
            gParser.GetValueRightToken(fVal, ',');
            gParser.GetValueRightToken(ptVal);
        }
        return TRUE;
        f.Close();
        delete [] szBuf;
    }
    CATCH( CFileException, e )
    {
        #ifdef _DEBUG
            afxDump << "File could not be opened " << e->m_cause << "\n";
        #endif
        delete [] szBuf;
        return FALSE;
    }
    END_CATCH
}
```

256 color Palettes in UniChat 2

Windows 256-color bitmap files have a palette in its header part. But in UniChat 2, we use only on color table for all the images except for the dialog. So we made a new type of data source for image files without palette. The master palette in UniChat 2 is defined as a static array in the Resource Manager object (CResMan). Then when we load each bitmap into memory, the color table entries are filled with this master palette values by calling CResMan::LoadMasterPalette(CDIB*).

```
static const PALETTEENTRY apeMASTER[256] =
{
#include "U2Pal.inc" // 256 Color Table
};
```

UDSGen removes the palette from each bitmap file.



The unified palette for our application is saved in CResMan source code.

When each bitmap file is loaded into memory from UDS file, the palette entries are filled by CResMan::LoadMasterPalette().

Predefined Palette Indices

In addition to the unified master palette of UniChat, some indices of the palette are assigned as a special purpose. For example, index 240 is registered as an outline color. Here is the rules for our master palette in UniChat.

| Index Range | Description |
|-------------|-------------------------------------------------|
| 0-9,246-255 | Windows System Colors (20) |
| 241-245 | Advertising clips (5) |
| 240 | Outline Color, rgb=(131,231,131) |
| 239 | Black (0,0,0) |
| 238 | White (255,255,255) |
| 237 | Outline Off, This is the current outline index. |
| 236 | Transparent Color |
| 232-235 | Hair Color Set #0 (4) - Image contains |
| 228-231 | Hair Color Set #1 - program switches |
| 224-227 | Hair Color Set #2 - program switches |
| 220-223 | Hair Color Set #3 - program switches |
| 216-219 | Face color set (4) |
| 212-215 | Clothes Color Set #0 (4) - Image contains |
| 208-211 | Clothes Color Set #1 - program switches |

| | |
|---------|-----------------------------------------|
| 204-207 | Clothes Color Set #2 – program switches |
| 200-203 | Clothes Color Set #3 – program switches |
| 196-199 | Fixed Color Set (4) |
| 10-195 | Background Colors (186) |

The reason to use these predefined sets of indices in drawing bitmap graphics is to give some variations in colors of the same image. For example, in a chat room if the same character is selected then the program automatically changes its hair and clothes color sets.

It's important for the graphic designers to keep this rule. They have to draw characters hair using only the colors in the hair color set #0. Other three sets for hair colors are reserved for our program to switch with the original indices.

```
void CResMan::RotateActorColorSet(CDIB* pDIB, const int nColorSet) const;
```

Every character has a border line or outline drawn with the index of 237. This rgb value is equal to the transparent color so users cannot identify the outline. But once the program determined that the character should show its outline, for the client's own character, when the program loads this image into memory, the rgb value of index 237 is replaced with that of index 240.

```
void CResMan::ShowOutline(CDIB* pDIB) const;
```

One important thing for our graphics system is that we use identity palette. We use this for better performance in animation which involves repeated rendering and drawing of the same image. In this scheme, while loading the bitmap image into memory, all the bits in the image are replaced with a new index set in the identity palette. So once we load an image from a bitmap file, we cannot say that the bits in the file are intact in memory.

How to make a dialog with 256-color image

There is no support for the 256-color bitmap in MFC. Using our own graphics library, UC2Ani, we can simply make a dialog window with a background image of 256 colors. Or you can tile the background.

Once you know how to use this graphics library, all the processes can be shown as a pattern. Here is the procedure. This code also shows how we can make a 256-color buttons.

| Step | Procedure |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Make a dialog template in resource view of the VC++ workspace panel. Using ClassWizard, make associated code and header file for the CDialog inherited class. Let's say it CTestDlg. |
| 2 | In header file add the following codes, <pre>#include "UC2Ani/DIB.h" #include "UC2Ani/DIBPal.h" #include "UC2Ani/PSButton.h" class CDIB; #define TESTFILE_BMP "c:\\test.bmp"</pre> |

In the class definition,

```
protected:
    CDIB*      m_pDIBBack;    // Background frame image
    CPalette*   m_pPal;       // main palette
    BOOL       m_bPaletteCreated;
```

3 In CMainFrame class, add a utility function to determine current video mode.

```
BOOL CMainFrame::Is256Palette() const
{
    BOOL bResult=TRUE;
    // Get a screen DC to work with.
    HWND hwndActive = ::GetActiveWindow();
    HDC hdcScreen = ::GetDC(hwndActive);
    ASSERT(hdcScreen);

    // Make sure we are on a palettized device.
    if (!::GetDeviceCaps(hdcScreen, RASTERCAPS) & RC_PALETTE)
    {
        bResult = FALSE;
    }
    else
    {
        // Get the number of system colors and the number of palette
        // entries. Note that on a palletized device the number of
        // colors is the number of guaranteed colors, i.e., the number
        // of reserved system colors.
        int iSysColors = ::GetDeviceCaps(hdcScreen, NUMCOLORS);
        int iPalEntries = ::GetDeviceCaps(hdcScreen, SIZEPALETTE);

        // If there are more than 256 colors we are wasting our time.
        if (iSysColors < 0 || iSysColors > 256)
        {
            bResult = FALSE;
        }
    }
    ::ReleaseDC(hwndActive, hdcScreen);
    return bResult;
}
```

4 In constructor,

```
CTestDlg::CTestDlg(CWnd* pParent /*=NULL*/)
: CDialog(CTestDlg::IDD, pParent)
{
    ...
    m_pPal = NULL; // Set it NULL before loading DIB

    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();

    CString strFile(TESTFILE_BMP);
    m_pDIBBack = new CDIB;
    if (!m_pDIBBack->Load(strFile))
    {
        delete m_pDIBBack;
        m_pDIBBack = NULL;
        return;
    }
}
```

```

    }

    if (pMF->Is256Palette()) // Assume that this function is already implemented
    {
        // Use mainframe's palette to avoid color flickering
        m_pPal = pMF->GetPalette();
        m_pDIBBack->MapColorsToPalette(m_pPal);
        m_bPaletteCreated = FALSE;
    }
    else // Use original palette in the file for TRUE color system
    {
        // Create the palette from the DIB.
        CDIBPal* pDIBPal;
        pDIBPal = new CDIBPal;
        ASSERT(pDIBPal);
        if (!pDIBPal->Create(m_pDIBBack))
        {
            AfxMessageBox("Failed to create palette from DIB file");
            delete pDIBPal;
        }
        m_pPal = pDIBPal; // type casting to parent class
        m_bPaletteCreated = TRUE;
    }
}

```

5 In destructor,

```

CTestDlg::~CTestDlg()
{
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal && m_bPaletteCreated)
        delete m_pPal;
}

```

6 Using ClassWizard add following message handlers:

```

virtual BOOL OnInitDialog();
afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
afx_msg BOOL OnQueryNewPalette();
afx_msg void OnSize(UINT nType, int cx, int cy);
afx_msg BOOL OnEraseBkgn(CDC* pDC);

```

7 Modify OnInitDialog()

```

BOOL CTestDlg::OnInitDialog()
{
    if (!m_pDIBBack)
        return FALSE;
    CDialog::OnInitDialog();
    m_btnOK.SubclassDlgItem(IDOK, this);
    m_btnCancel.SubclassDlgItem(IDCANCEL, this);

    CPoint ptLT(349, 238); // find adequate button position
    m_btnOK.MoveResize(ptLT);
    ptLT.x = 17;
    m_btnCancel.MoveResize(ptLT);

    return TRUE; // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}

```

8 Add palette message handlers.

```

void CTestDlg::OnPaletteChanged(CWnd* pFocusWnd)
{
    CDialog::OnPaletteChanged(pFocusWnd);

    if (pFocusWnd != this)
        OnQueryNewPalette();
}

BOOL CTestDlg::OnQueryNewPalette()
{
    if (m_pPal)
    {
        CDC* pdc = GetDC();
        CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE);    // foreground
        UINT u = pdc->RealizePalette();
        if (pPalOld)
            pdc->SelectPalette(pPalOld, FALSE);
        ReleaseDC(pdc);
        // if (u)
        // { // Some colors changed so we need to do a repaint.
        //     Invalidate(); // Repaint the lot.
        //     return TRUE; // Say we did something.
        // }
        return FALSE; // Say we did nothing.
    }
}

```

9 Finally add a handler for WM_SIZE to fit for the background image.

```

void CTestDlg::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);

    if (m_pDIBBack)
    {
        SetWindowPos(NULL, 0, 0,
                     m_pDIBBack->GetWidth(), m_pDIBBack->GetHeight(),
                     SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
    }
}

```

MFC SDI Architecture

It's very helpful if you know the sequence of calling MFC functions in your MFC SDI application.

Loaded symbols for 'C:\WINDOWS\SYSTEM\MSVCRTD.DLL'

Loaded symbols for 'C:\WINDOWS\SYSTEM\MFC42D.DLL'

// Loading...

CGenSDIApp::CGenSDIApp() // Application class constructor

CGenSDIApp::InitApplication() // for backward compatibility

CGenSDIApp::InitInstance()

```

CGenSDIDoc::CGenSDIDoc() // Document constructor

CMainFrame::CMainFrame() // Frame Window constructor
CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
CMainFrame::PreCreateWindow(CREATESTRUCT& cs) // why called twice?
CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct) // WM_CREATE
CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)

CGenSDIView::CGenSDIView() // View Window constructor
CGenSDIView::PreCreateWindow(CREATESTRUCT& cs)
CGenSDIView::OnCreate(LPCREATESTRUCT lpCreateStruct) // WM_CREATE

CGenSDIDoc::SetTitle(Untitled)
CGenSDIDoc::DeleteContents()
CGenSDIDoc::OnNewDocument()

CGenSDIView::OnInitialUpdate()
CMainFrame::ActivateFrame(int nCmdShow)
CGenSDIView::OnDraw(CDC* pDC=0x63f970)

// Application Logic will be here.

// Closing...
CMainFrame::OnClose() // WM_CLOSE

CGenSDIDoc::CanCloseFrame(CFrameWnd* pFrame=0x750d14)
CGenSDIDoc::OnCloseDocument()

CMainFrame::OnDestroy() // WM_DESTROY

CGenSDIView::OnDestroy() // WM_DESTROY
CGenSDIView::~CGenSDIView() // View Window destructor

CMainFrame::~CMainFrame() // Frame Window destructor

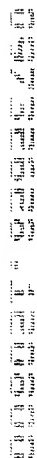
CGenSDIDoc::DeleteContents()
CGenSDIDoc::~CGenSDIDoc() // Document destructor

CGenSDIApp::ExitInstance()

```

The program 'D:\MsDev\Projects\kSm\GenSDI\Debug\GenSDI.exe' has exited with code 0 (0x0).

7. Class Hierarchies



APPENDIX C

GRAPHIC CHATTING WITH ORGANIZATIONAL AVATARS

KYUNAM KIM

M-8786 US

[illegible]



Unichat Networks Inc. (UNI) Business Proposal



McDonald's New Virtual "McWorld"

Written by Unichat Networks Inc. (UNI), October, 2000
Copyright©, preliminary proposal
This is not an exclusive agreement
All rights reserved for UNI

McDonald's, McWorld, its products and characters mentioned are
All Trademarks registered to
McDonald's Inc.©

Contact:
Ken Kim
Unichat Networks Inc.
627 National Avenue
Mountain View, CA 94043
Tel: 650.961.0292
www.unichat.com

Executive Summary

UNI is offering McDonald's its unique running technology, proven to be effective in connectivity and interactivity, to further develop and prepare McDonald's advertising and marketing future. Eventually, all businesses will have to take at least some part in on-line business. The question is how will McDonald's make its site more attractive and user friendly to its target customers, be better than its competition, and most of all, cut costs while increasing profit. We believe that Unichat is McDonald's best solution. Unichat can provide:

- Electronic Customer Interaction Management (ECIM)
Interactivity between McDonald's and customers; customers and customers.
- Electronic Customer Relationship Management (ECRM)
Connectivity which provides and receives important information/feedback instantly and easily.
- Increasing the reach of the advertisements: McDonald's Globalization.
Whereas McDonald's stores' job is to sell the hamburgers, fries, and nuggets, Unichat-developed McWorld is to sell McDonald's itself, the name, the image, turning intangible assets into a profitable business.
- Reaching the crucial teen market: providing a stimulating, physical, personal space for teens - the best way to connect with them and win their business. At the same time, McDonald's can get informed of what interests them. McDonald's can give teenagers reasons that teenagers need, more than great deals on food, to develop a far greater loyalty to the McDonald's name beyond time, space and material. With Unichat, McDonald's can have the same reputation among teenagers as it already has with young children.

The bottom line is increased profit through more channels and a futuristic company image for its current customers and future ones.

UNI is asking McDonald's to evaluate its advertising and marketing spending and to see that Unichat-developed McWorld can cut advertising and marketing costs while reaching many more people. Users will be communicating through characters, actions and sounds that are universal applicable that anyone and everyone around the world will be able to interact with McDonald's = One, interactive, connective, relationship building and influential tool for the entire world.

For these reasons, UNI proposes licensing Unichat technology for McDonald's

Table of Contents

| | | |
|-------|-----------------------------------------------------------|-------|
| I. | What is Unichat : | |
| | a. Brief UNI Background | |
| | b. Unichat Technology | |
| | c. Unichat's Ability..... | 4 |
| II. | Why McDonald's? : | |
| | a. Established World Wide Brand of Quality | |
| | b. Well known Connection with Children, what about teens? | |
| | c. Global Community Building | |
| | d. Future Outlook: Get Ahead!..... | 5,6 |
| III. | Why Unichat for McDonald's? : | |
| | a. Socializing on the Internet | |
| | b. Share Key Business Characteristics | |
| | c. Cutting Edge Interactive Environment | |
| | d. Expand Ad-Market | |
| | e. Gain Advantages of On-line Business..... | 7,8 |
| IV. | Implementation of Unichat = Virtual "McWorld" | |
| | a. Character Based Online Community | |
| | b. Target Audiences | |
| | c. Sample Scenario..... | 9 |
| V. | Functions : Something for Everyone | |
| | = All Connected under "McWorld"..... | 10 |
| VI. | Results : The Bottom Line | |
| | a. What McDonald's Gains.... | |
| | =Satisfied and Loyal Customers/Friends..... | 11 |
| VII. | Future Implementations..... | 12 |
| VIII. | The First Step Together: Estimated Cost and Time | |
| | a. Phase One | |
| | b. Phase Two | |
| | c. Phase Three..... | 13,14 |

I. About UNI and Unichat:

a. Unichat Networks Inc.

First established in the United States in 1999, UNI has since also been established in Brazil and Japan. For UNI, Unichat's chatting business is only the beginning. UNI's global vision includes Uniplay, a game solution applicable to the latest CD-ROM and on-line game systems; PDA solutions for personal systems such as the Palm Pilot; and a mobile, wireless phone solution. Unichat currently has a running site www.unichat.com.

b. Unichat Technology

Unichat uses a pioneering graphic character chatting technology using cartoon-like, animated avatars in 3-D Settings simulating a live, interactive environment. The avatars resemble colorful human and animal characters that are moving, active, and controllable by users who choose these to represent themselves on-line. They are used to communicate, display various actions/gestures and express emotions thereby simulate the next best human interactivity. For example, avatars can cry, laugh, make musical sounds, punch, disappear, morph into other avatars and of course talk through cartoon-like bubbles. Controlled with mouse and/or arrow keys, the techniques and rules of engagement are easily learned, allowing creative freedom. They exist in infinite number of standard settings and provide the very tools for the users to fully create their own "rooms," that range from castles to recreation parks.

c. Unichat's Proven Ability

Unichat offers a gaming and community atmosphere where people continue to come back to for entertainment and interaction. The high-tech program is easily downloadable, installable with CD and operates on all P.C.'s without any special applications. Its user-friendly and unique features attract all types and ranges of people simultaneously through one system. Unlike non-graphic chat services, users of Unichat do not actually have to chat. The avatars are fun and exciting to play with in and of itself that users are on-line without chatting, longer and more frequently. The uniqueness of Unichat encourages consistent loyal users.

By opening the online-customer-interaction doors, Unichat can effectively provide Electronic Customer Interaction Management (ECIM), as well as Customer Relationship Management (ECRM) solutions.

II. Why McDonald's?

a. Established World Wide Brand of Quality

UNI believes that McDonald's is the right company for this joint venture for many reasons. Number one is that McDonald's already has an established strong brand name, which Unichat believes is extremely advantageous in crossing into on-line world. Almost every single person in the U.S. is guaranteed to have visited McDonald's at least once and everyone has a friend who worked at McDonald's. McDonald's is an established household brand name/image existing in the minds of all Americans for its continuous quality in product, performance and service, more than any other fast-food chain in America, perhaps the world. McDonald's name connotes a trust and a tradition in the eyes of the public.

Therefore using what McDonald's as already accomplished, UNI believes that McDonald's can successfully carry their popularity to this new venture and allow McDonald's the best on-line business start possible.

b. *Well Known Connection with Children, what about teens?*

McDonald's, unlike their competitors, also have established trademark characters who represent the store and the company. These characters are also universally recognized and have become household names for children who are excitedly entertained by these characters. Also with McDonald's-only co-promotion with other famous children products, children have come to naturally connect McDonald's with fun. As explained earlier, Unichat consists of using similar themes to attract this age group. UNI believes that McDonald's characters can be incorporated into Unichat technology to create a safe, interactive and educational on-line access for children and strengthen McDonald's tie with them.

Once these McDonald's children are all grown up, how will they connect or associate with McDonald's? Teens are an important group to target because when they come to McDonald's, it is because they themselves are choosing to, more so, than young children; and they are also major consumers of fast foods. Then teens will soon grow up to be young adults and parents themselves with influence on their children. Thus teens right now, need their own connection to McDonald's, other than just by McDonald's foods to make them loyal McDonald's friends.

c. *Global Community Building*

McDonald's goals include building more McDonald's communities around the world, especially Asia, as stated in the official McDonald's web site. Having already achieved much success in the United States, UNI believes that the future growth of McDonald's rests in its performance in other countries where

there are bigger rooms to grow. There is no better, easier way for such an expansion than using the Internet, better yet, Unichat. Studies of trends have shown that in Asia, Internet, and chatting, especially, is one of the most popular

ways of conducting everyday life. Also, the psychological and emotional nature of chatting invites everyone and anyone, suiting the entire world. Soon, every business, regardless of product will have to merge into an on-line business. Presently, McDonald's websites do not seem to be in line with what UNI believes the virtual McWorld can be.

d. Future Outlook: Get Ahead!

McDonald's has undoubtedly maintained its leadership in the fast-food industry. But the future of McDonald's is not limited to its food with what and how is still up for grabs. UNI can help sustain this leadership with the changing needs and times of the people.

UNI believes the above McDonald's qualities makes the company ripe for a whole new level: developing and maintaining a relationship with people not only as customers, but also as loyal friends.

III. Why Unichat for McDonald's?

a. Socializing on the Internet

Internet is the next and continuously growing sector, which McDonald has not yet explored in full. The success of Internet businesses is that customers are participants who have choices, voices and unlimited accessibility. Those characteristics are exactly what make Unichat popular. Through its innovative and exclusive technology, Unichat has set a new trend in cyberspace interactivity, reaching and satisfying customers. Socializing at McWorld is just as significant as friends meeting at McDonald's for lunch. McWorld can be a replica of the actual experience of visiting a McDonald's store, but much more.

b. Share Key Business Characteristics

Same targeted audience

The customers that McDonald's target its advertisements are primarily, children, adolescents and teenagers. This is the very audience who are most intrigued by Unichat services and the same audience who participate on our site.

Common Aim of Advertising

McDonald's uses cartoon characters of TV shows, and movies, toys of famous brands, famous teen idols, athletes and games. Unichat uses cartoon style avatars with physical and "real" abilities in graphically enhanced environments for customers to communicate, interact and have fun with others. What the two have in common is that both McDonald's and Unichat aim at tapping into the popular culture of children and teenagers.

We have researched what this audience wants and we believe Unichat services are the best way to attract and maintain this audience online most frequently and for the longest time. Teenagers all around the world especially want interactivity, semi-anonymity, variety, quick and easy fun. Unichat accommodates each of these needs creatively and more effectively than other typical on-line services or advertisements. Ultimately with the cooperation between UNI and McDonald's, McDonald's can gain more of the audience they want without having to rely solely on co-promotion (extra cost) or sale on food (cut profit).

c. Cutting Edge Interactive Environment

UNI is offering an opportunity for McDonald's to achieve a greater company vision unprecedented by other fast food or chain restaurants. Unichat's technology will create an interactive image well-suited and most advanced for McDonald's off-line reputation.

d. Expanded Advertisement Opportunities

Advertisement can take place in McDonald's own time in the cyber world instead of designated, limited space and time on TV or Radio, while their best qualities are not compromised but enhanced.

Unlike limited advertisements options on-line like having only banner ads, or only sponsorships, McDonald's can have the most unlimited and variety of customizable and expandable types of advertisement. Also, at present, McDonald's advertisements are presently catering primarily to off-line, fast food customers. With Unichat implementation, McWorld can represent and advertise McDonald's regardless of their fast-food preference.

With UNI, advertisement is not so straightforward anymore. So McDonald's will no longer be viewed primarily as a "product selling" company. McDonald's will be viewed as a service provider, allowing people's connection with McDonald's to move beyond fast food, and stunning its customers and its competition with its innovative new direction. UNI is offering a business solution, which is like no other marketing tool and cannot be like any other because it will be specially customized for McDonald's. With Unichat, McDonald's is taking a step towards a new future.

e. Gain Advantages of Online Business

UNI can show that McDonald's is a multi-faceted company interested in feeding a variety of people's needs where developing new ideas and implementations is only a matter of time. Through Unichat's technology, language and culture will no longer matter. One medium can attract and accommodate a universal audience.

Unichat believes that this technology will help McDonald's expand its advertising market by tapping into on-line business mechanisms, maximize advertisement effectiveness and thereby increase its profit growth.

IV. Implementation of Unichat = Virtual McWorld

a. Character Based Online Community

Using Ronald McDonald and other McDonald's trademark characters, products and other recognized McDonald's settings, Unichat's Technology can be used to create an on-line world of McDonald's for McDonald's customers, patrons, employees and co-promoters. Customers can reach this "community" through McDonald's official website and or/ by CD's distributed through various channels such as Happy Meals, or other promotional events.

b. Targeted Audiences

One medium -- numerous benefactors

- Customers/Friends of all ages:
Children; Teenagers; Young Adults; Parents
- Employees
- Charity
- Sponsors/Co-promoters
- McDonald's

c. Sample Scenario:

- Customized setting: colorful, stimulating, active backdrop and props of familiar McDonald's characters and store features. A virtual recreation of any desired McDonald's indoor or outdoor environments including: Restaurants, theme parks, concert halls, corporate offices, university classrooms and imaginary locales.
- Environments populated with McDonald's characters like: Ronald, Grimace, Hamburgler and Birdie. Users take on a virtual likeness chosen from over 40 characters (male, female, old, young, multi-races and even animals)
- Equipped with appropriate language and behavior censors and filters.
- The room full of banner advertisements of McDonald's products.
- Click-on high quality games featuring McDonald's characters to win coupons or tickets with a store locator indicating the closest store where one can redeem the coupon.
- All of the visual contents can be downloaded and/or printed out by the user.
- Info link, promos with other toys link, game links for different ages including one for parents.
- Themed rooms for discussion, entertainment, etc.
- Connect with a family or friend at RMH.
- Suggestion box for McDonald's.
- Live chat with celebrities in specialized settings. Custom celebrity avatars can be created!

V. Functions: Something for Everyone

- Advertising banners with the latest sales and events just like TV ads of McDonald's as well as other companies of McDonald's choosing.
- New Ronald's Play House with involving games, contests, and etc. for children and teenagers
- Chatting with themed rooms, which can be specialized according to the type of research McDonald's would like information on. Special "rooms" for holidays, events, sports, etc.
- Safe and Educational Internet for toddlers and young children who cannot "chat": Happy Meals can feature customized CD's containing the basics of Computer/Internet education to be used with parents.
- Providing ways customers can purchase or win promotional prizes: Redeemable coupons and tickets can be won on-line, printed up on their own computers, to be used at stores. Cuts production and assembly cost
- Host charity events on-line using the games: new ways to raise money for the Ronald McDonald House making participation easy and fun for everyone Empowering users
- Families and sick children staying at RMH can use the site to communicate with each other. RMH and what it is about and how people can help is more inviting
- Can hold employee meetings, forums, anonymous feedback and discussion. Employee only rooms where employees around the country and globe can communicate with each other: boosting employee morale and company spirit.
- Chat with contracted celebrities: Olympic athletes, music groups N'Sync®
- Collaboration with featured cartoons and toys such as Pokemon® and customize Pokemon® avatars to be used to chat or play games; advertisements of participating movie or product premier dates and information.

Unichat can connect everyone under McDonald's umbrella, McWorld!

VI. Results: The Bottom Line

- Lower advertisement/marketing costs: this promotional vehicle is less expensive than the “hard goods” McDonald’s usually uses (i.e. action figures, beanie babies) while attracting the same type of excitement for McDonald’s.
- 24-hours of advertisement. Customizable and adjustable, conveniently and quickly.
- Wider range of audience/participants in age, location, interests and time.
- Can be a corporate leader in providing children’s first basic education about computers and the Internet usage.
- Development of trusting relationship with parents.
- Accessibility of McDonald’s for its customers/friends on their own personal operating time.
- Broadening its reputation from the leading fast-food maker and distributor to an entertaining and educating community, a way of life.
- The first fast-food chain to do so: gain a competitive advantage
- Receive instant, valuable feedback from users on anything from McDonald’s products to future directions of the market, and demographic marketing data in real time environments. (i.e. surveys, polls on new product, customer service, etc.)
- Easy, universal communication mechanism for all McDonald franchise stores and employees.
- A McDonald’s community and loyal friends can develop even in locations without a McDonald’s store.

Unichat can establish satisfied and loyal customers/friends from childhood, for life!

VII. UNI's Future implementation ideas for McWorld...

Expansion of McDonald's On-line business:

- ◆ Sale or auction - of collectible McDonald's Happy Meal Toys
- ◆ E-commerce - of McDonald's character-based promotional materials and virtual order processing for almost any product
- ◆ Virtual Reality - Sale of time and space on McDonald's site
- ◆ Virtual McDonald's University – Online virtual training and distance learning with a live, graphic interface
- ◆ Mobile McDonald's Land – hook ups to mobile wireless devices such as cell phones and wireless PDA devices. Activities, coupon generation and Global positioning, remote restaurant locator services.



The possibilities for *Virtual McWorld* are endless. As the corporate vision continues into the new century and as technology continues to grow, Unichat can easily scale its operation to stay always ahead of the pack to provide worldwide quality of interactive activities, distance learning, and marketing opportunities.

VIII. The First Step Together; Estimated Costs and Time

| | Phase One | Phase Two | Phase three |
|--------------------|-------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| 1) Function | Unichat base | McDonald's base | McWorld, same as McDonald's fast-food store |
| | License Fee: \$500,000 | MapEd License Fee: \$500,000 | Upgrade Fee: \$200,000 |
| 2) Room | Unichat Room with McDonald's tile | 20 customized McDonald's Room | McDonald's Room Plus Sponsors' or Co-promoters' Rooms |
| | New Tile in 50 Rooms: \$100,000 | 20 rooms: \$400,000 | Paid by Sponsors/Co-promoters=McDonald's Profit |
| 3) Character | Unichat Character with 1 McDonald's Character | 10 McDonald's unique characters | 10 McDonald's character with Sponsor's Character: ex. Ty animal |
| | New Character with 44 frames: \$25,000 | 10 characters with 44 frames: \$250,000 | Paid by Sponsors...=McDonald's Profit |
| 4) Games | Get coupons for Free drinks, or get Unichat characters: simple. | McDonald's promotion games like Mcpuzzle, Catch BigMac: Single Player Games | McDonald's Promote with game sponsors like Sony, Nintendo, etc.: Multi-Player Games |
| | New game: \$270,000 (Publishing not included) | 2 McDonald's games: \$800,000 (Publishing not included) | Paid by Sponsors...=McDonald's Profit |
| 5) Concurrent user | Under 5000 | Under 10,000 | More than 50,000 |
| 6) Hosting | Unichat 3 Computers for Chat, Web and Data servers T1 line for 5 Months | Using Co-Locations 5 computers T1 plus 5Mbps co-locations service. Ex.: (above.net) | McDonald's 9 Computers T1 plus 20Mbps or T3 Computer room |
| | Cost H/W: \$150,000 | Cost H/W: \$400,000 | Cost H/W: \$1,300,000 |

Continued on next page

| | Phase One | Phase Two | Phase three |
|-------------------------------------------------------|---------------------------------------------|--------------------------------------------------------|--------------------------------------------------------------------|
| 7) Advertising | McDonald's Tile | McDonald's Tile, Full Screen, and Banner | Sponsor's Banner |
| | No extra cost | Reduced Marketing Development Fund (MDF) | Paid by Sponsors...=McDonald's Profit + Reduced MDF |
| 8) ECRM (Electronic Customer Relationship Management) | Database, Requested by McDonald's | Customer service and Employee Service; Market Database | Marketing Research and Simulation. Ex: What Teens want? |
| | Cost S/W: \$150,000 | Upgrade S/W: \$150,000 | Upgrade S/W: \$150,000 |
| 9) Others | Language/Content Filtering Censor \$120,000 | Greeting card \$350,000 | Multi players McDonald's game \$1,500,000(publishing not included) |
| 10) Schedule (D day is Starting day) | D day + 6 Month | D day +12 Month | D day + 18 Month |
| Total Cost | \$1,315,000 | \$2,850,000 | \$3,150,000 – Income from Sponsors/Co-Promoters |
| *Remarks | -Maintenance charge will be included | | |

Unichat's proposed, new virtual McWorld, is highly multi-functional, forward looking, and best of all, profit generating in many ways :
The best business solution for McDonald's